Cloud Computing

Virtual Resources and Distributed Cloud Applications

Chapter 2: Virtual Resources

Distributed and Operating Systems
Electrical Engineering and Computer Science
Technische Universität Berlin

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Challenges for IaaS Provider

- Rapid provisioning
  - Resources must be available to the consumer quickly
  - No human interaction during provisioning
- Elasticity
  - Create illusion of infinite resources
  - Yet, manage data center in a cost-efficient manner
- Isolation of different consumers
  - Users must not interfere with each other
- Performance
  - Maintain good performance despite other challenges

Approach: Virtualization

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand **network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)** that can be rapidly provisioned and released with minimal management effort or service provider interaction.
>
> Standard 800-145

- Main idea on IaaS-level: provide virtual resources
  - Provide compute resources in the form of virtual machines or containers
  - Also, other computing resources such as virtual networks, virtual network functions, distributed storage (e.g. distributed file systems or block storage)

What is Virtualization?

- Definition of virtualization according to NIST:

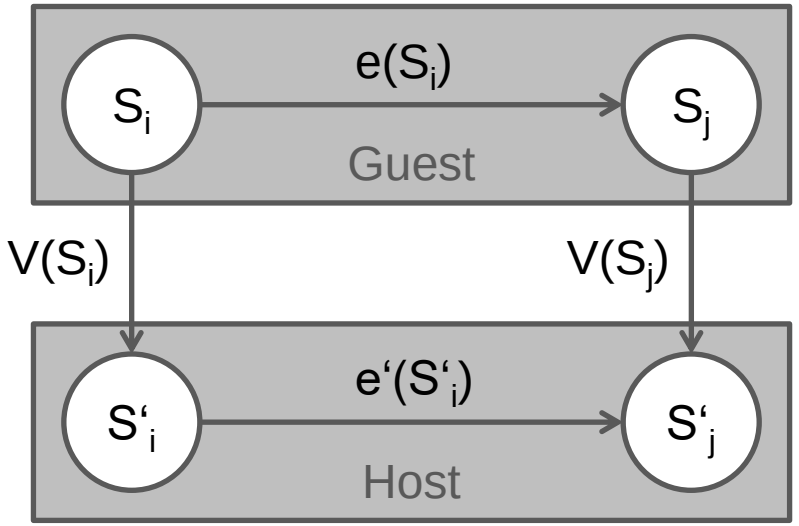
> "Virtualization is the simulation of the software and/or hardware upon which other software runs. This simulated environment is called a virtual machine (VM)."

- Virtualization can transform a real system so
  - it looks like a different virtual system
  - multiple virtual systems

- **Real system** is often referred to as **host (system)**
- **Virtual system** is often referred to as **guest (system)**

Formal Definition of Virtualization

- Isomorphism V:
  - Si, Sj: States of machine
  - e: Sequence of operations



- Isomorphism V maps guest state to host state such that
  - for e that modifies the guest's state from S_i to S_j
  - there exists a corresponding sequence of operations e' that performs an equivalent modification between host's states (S'_i to S'_j)

Virtual Machines (1/2)

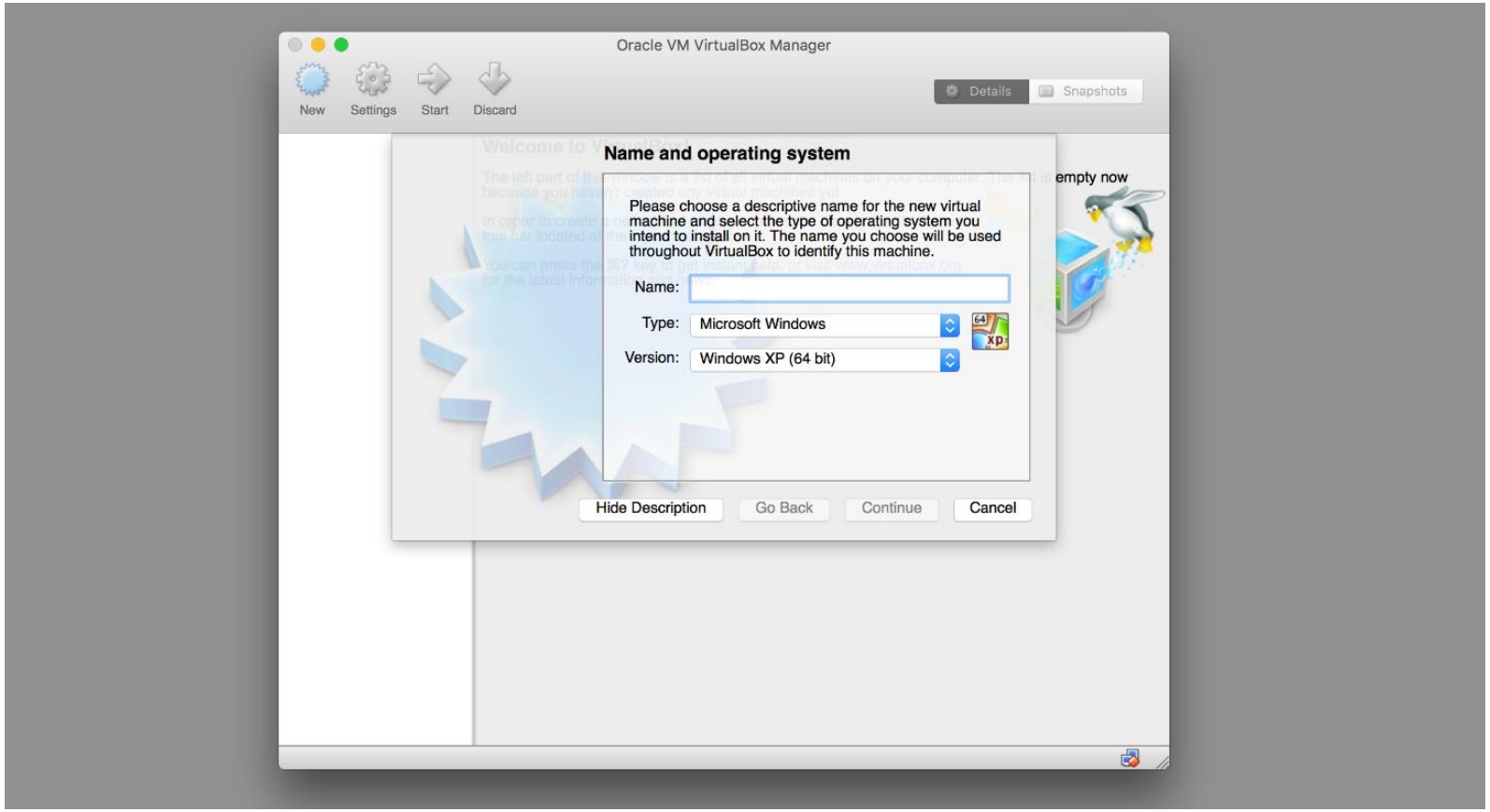
- Virtual machine (VM): hardware that is used by operating system and running processes is not real, but instead virtual → OS and processes run on software emulating hardware
- Use cases:
  - Run a different operating system than installed on host
  - Run applications in isolation
  - Run multiple virtual machines on a single physical host (pooled resources for higher utilization and lower cost)
- VMs often provided by Infrastructure-as-a-Service clouds, e.g. public clouds like Amazon's EC2

Virtual Machines (2/2)

- VMs can have different hardware characteristics (including different from the physical host)
  - e.g. number of cores, assigned memory, disk space
- VMs are often started from available images (with an operating system and some applications installed)
  - e.g. Windows with SQL Enterprise
- Public cloud providers charge depending on VM type and usage time
  - Predominant model: “pay by the hour” of VM instances
  - Also: pay-per-use of specific resources (e.g. data downloaded from VM instances)

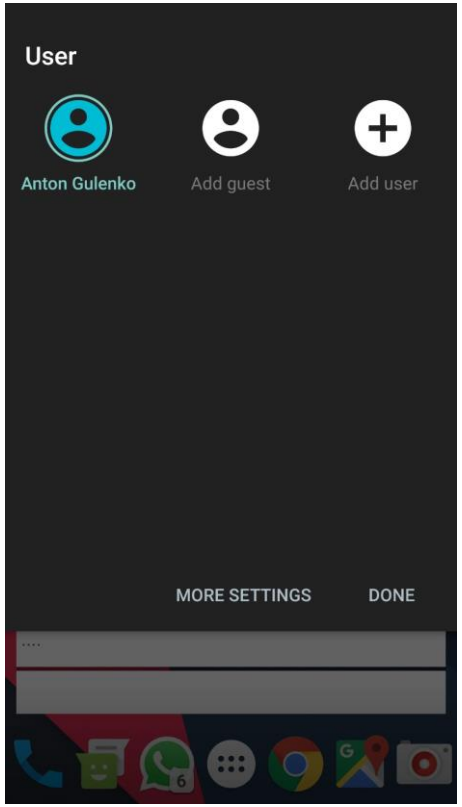
Use Cases for Virtual Machines (1/4)

- Run a different operating system than your host operating system



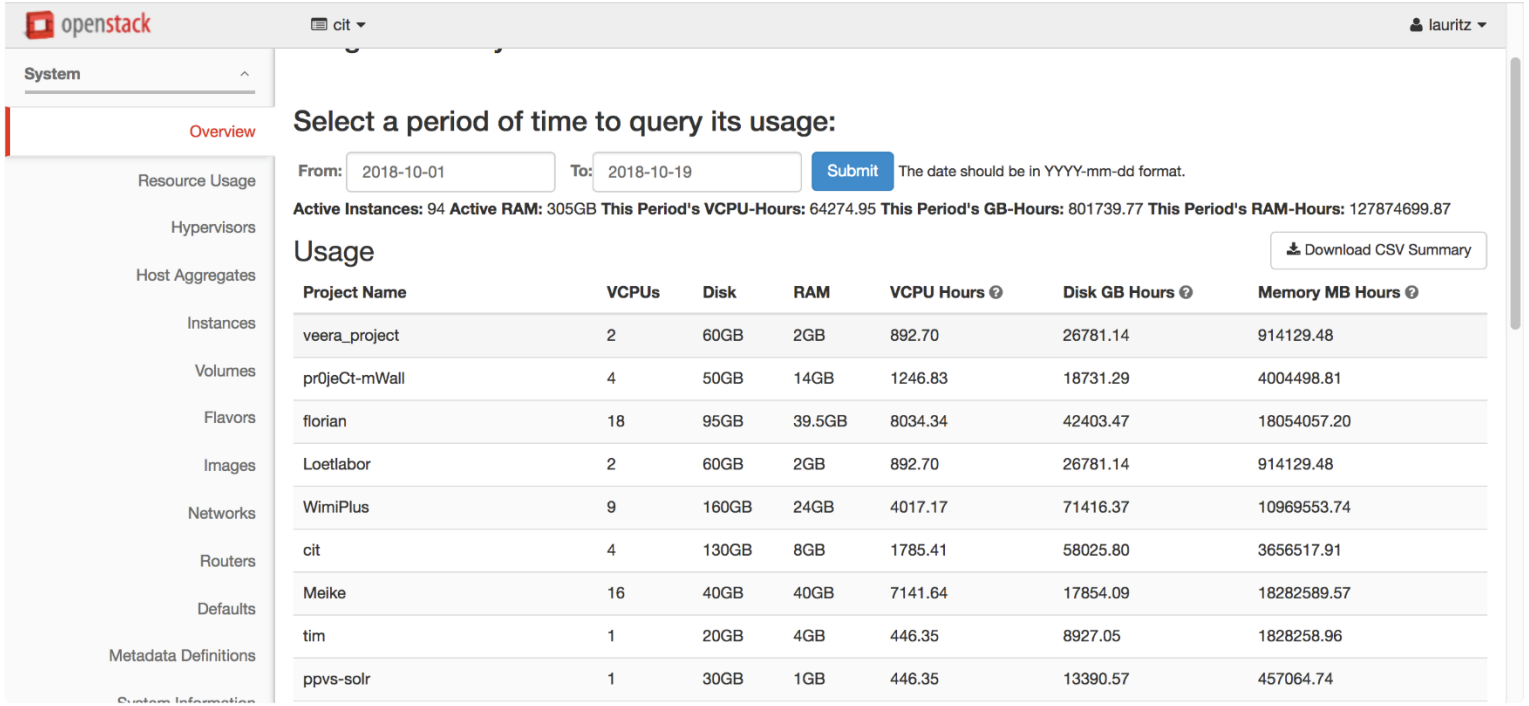
Use Cases for Virtual Machines (2/4)

- Operating multiple isolated environments on a host (e.g. different user accounts on a phone)



Use Cases for Virtual Machines (3/4)

- Pool of resources shared by multiple users and applications in a private cloud



Use Cases for Virtual Machines (4/4)

- Virtual machines available in a public cloud (AWS)

Linux | RHEL | SLES | Windows | Windows with SQL Standard | Windows with SQL Web

Windows with SQL Enterprise | Linux with SQL Standard | Linux with SQL Web | Linux with SQL Enterprise

Region: EU (Frankfurt)

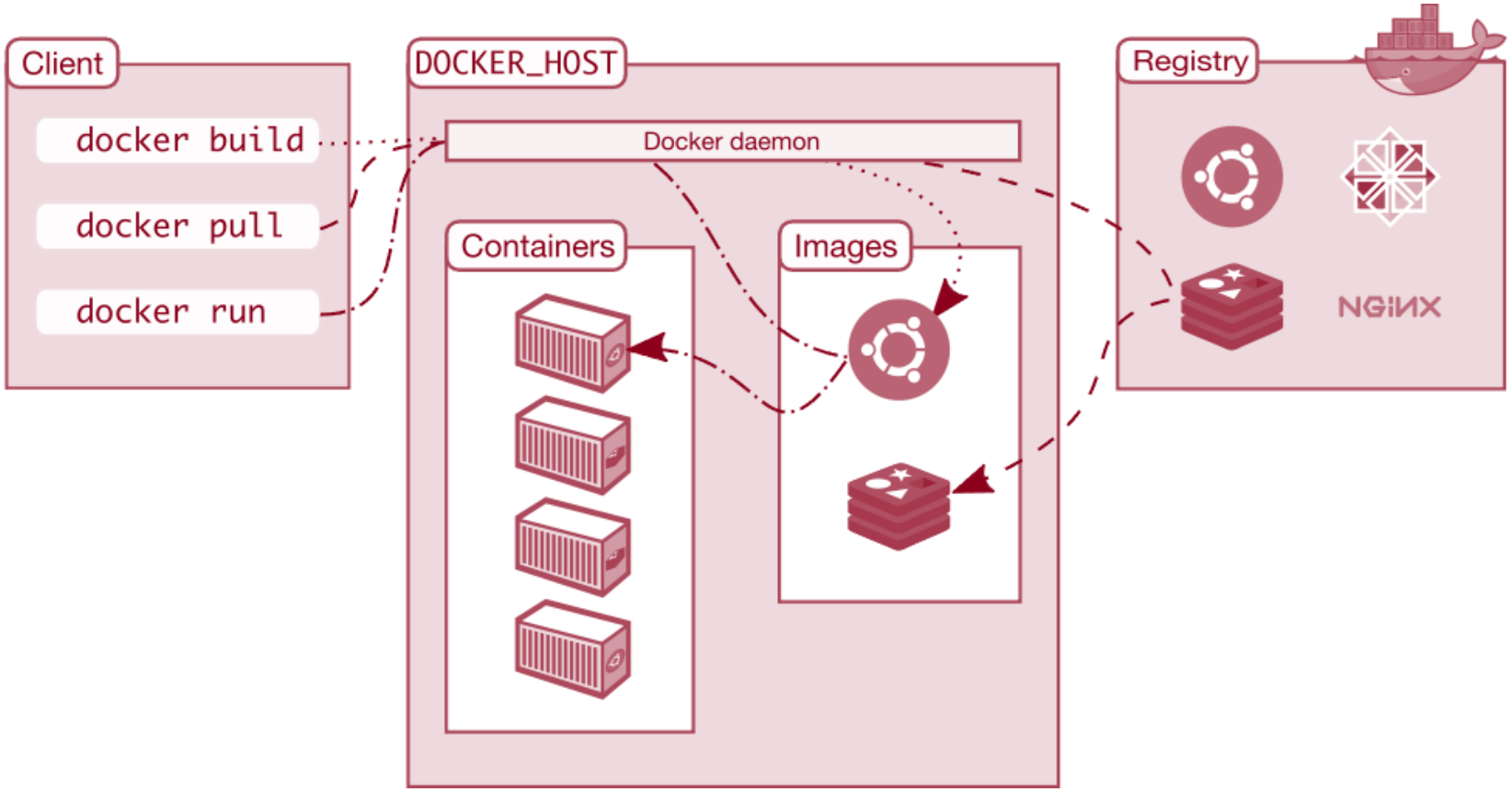
	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t3.nano	2	Variable	0.5 GiB	EBS Only	$0.006 per Hour
t3.micro	2	Variable	1 GiB	EBS Only	$0.012 per Hour
t3.small	2	Variable	2 GiB	EBS Only	$0.024 per Hour
t3.medium	2	Variable	4 GiB	EBS Only	$0.048 per Hour
t3.large	2	Variable	8 GiB	EBS Only	$0.096 per Hour
t3.xlarge	4	Variable	16 GiB	EBS Only	$0.192 per Hour
t3.2xlarge	8	Variable	32 GiB	EBS Only	$0.384 per Hour
t2.nano	1	Variable	0.5 GiB	EBS Only	$0.0067 per Hour

Containers

- Lightweight “OS-level virtualization”: virtual environments for single applications → processes run in isolation within an OS
  - Containers rely on containment features of the OS kernel (e.g. the *chroot* and *cgroups* features of Linux)
- Use cases:
  - Bundling dependencies and the environment of applications (libs and config) to ship software
  - Isolating applications on the same host from each other (→ resource sharing/pooling in a cluster)
- Containers can have different sizes, are usually started from images, and can be run on cloud services

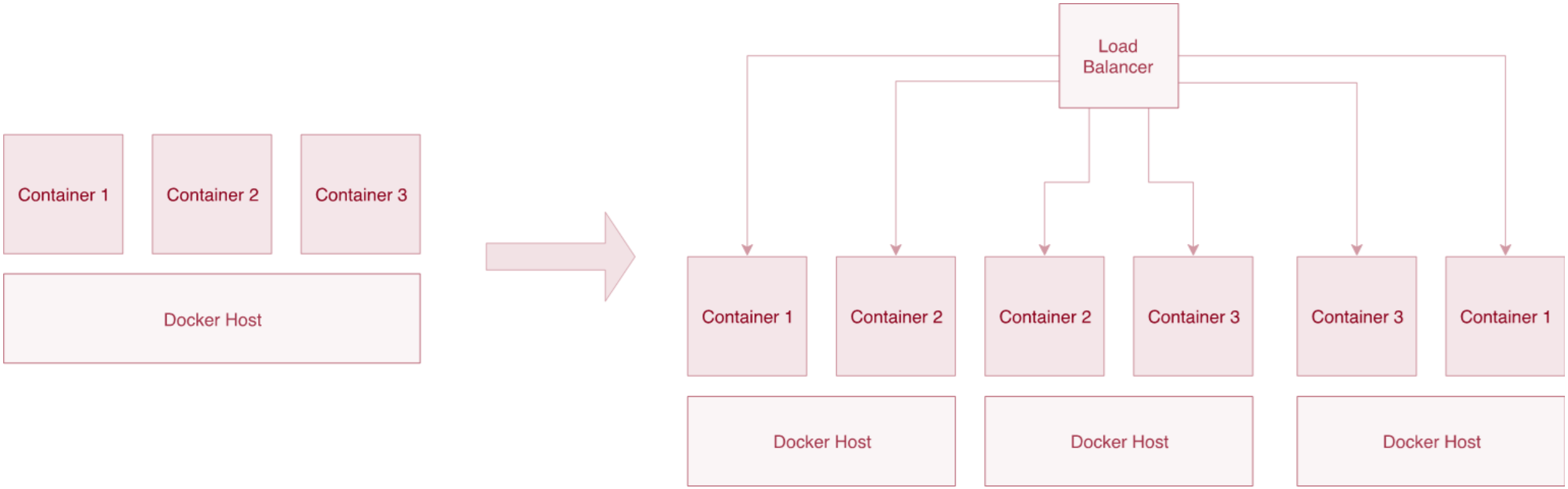
Use Cases for Containers (1/2)

- Build and ship your application in a container to be able to run it everywhere, e.g. development on your host, but then deployment to staging / production



Use Cases for Containers (2/2)

- Having many applications share the resources of a cluster



Containers vs. Virtual Machines

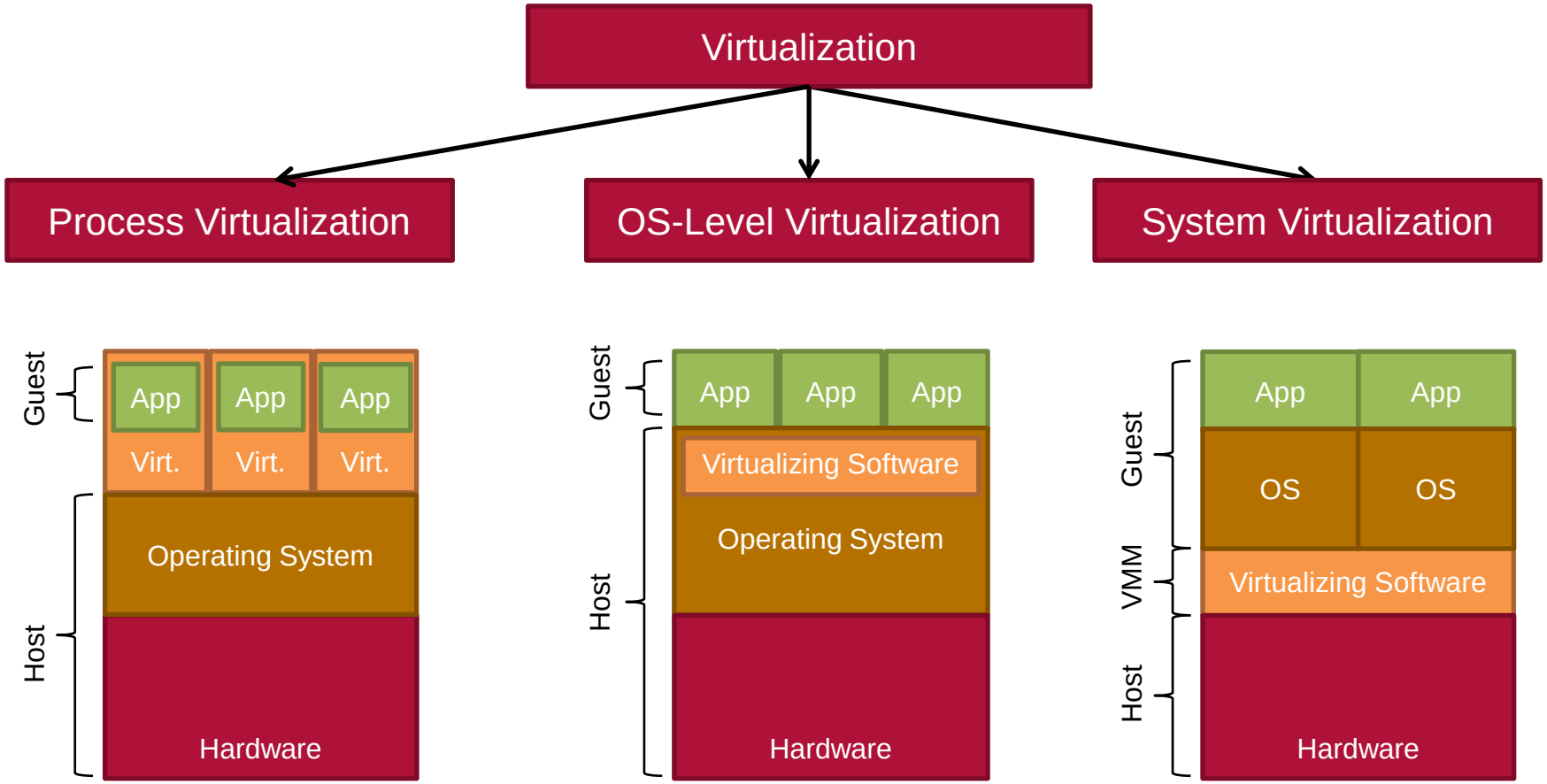
- Containers
  - No virtual hardware: cannot run an application build for a different architecture or operating system kernel
  - Reduced scope: single application in pre-build environments
  - Reduced isolation: containerized applications share the same kernel, but are isolated on process-level
  - Live migration of containers is less mature

- VMs
  - Larger images, slower VM startup, and more overhead per VM on a physical node

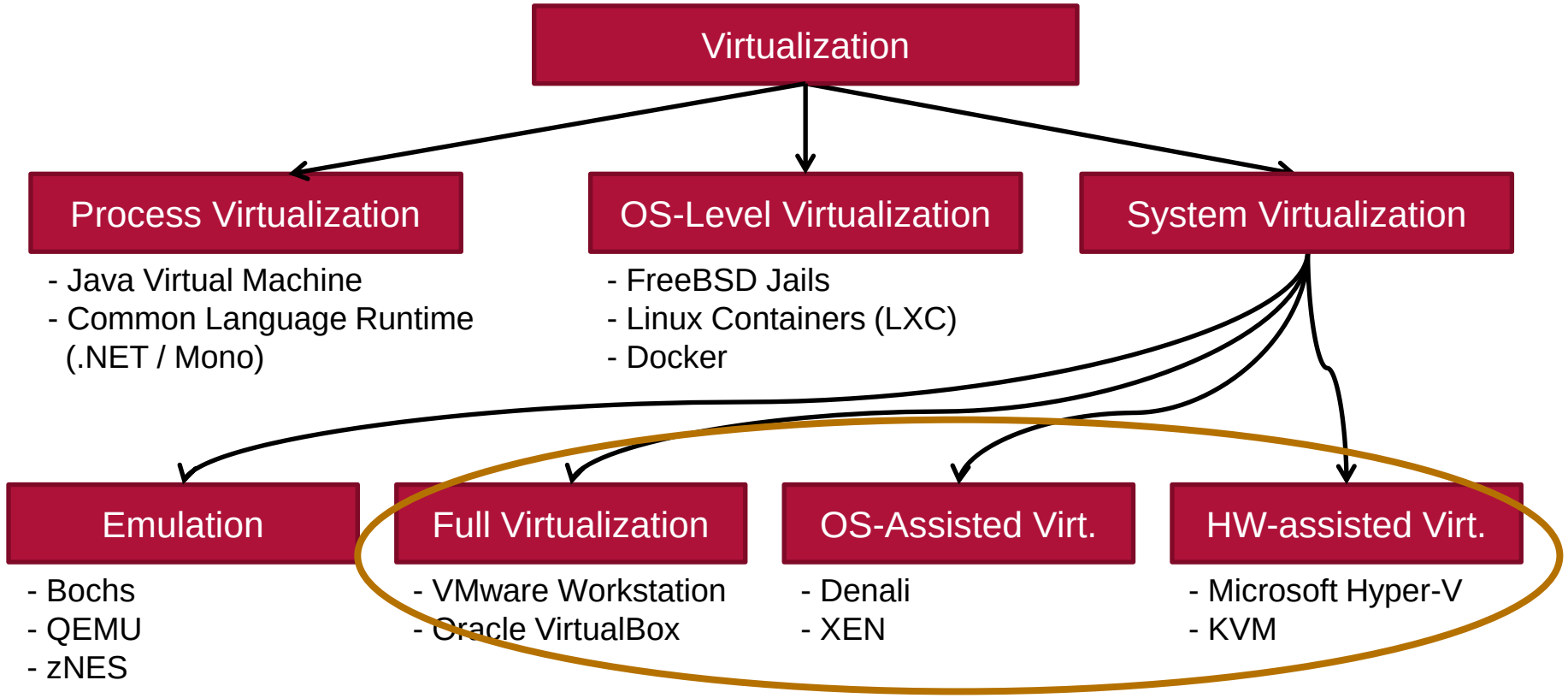
Does Virtualization Solve the Provider‘s Challenges?

Rapid Provisioning	Elasticity	Isolation among consumers	Performance
✓ -VM instances can be started from pre-compiled images -VM images can be stored as logical volumes on SANs or simply as files	✓ -Resource sharing creates illusion of unlimited resources to single customer -Important: Customer has incentive to release idle resources	? Open questions: - What degree of isolation can virtualization really provide? - Are resources distributed in a fair manner?	? Open questions: - What is the overhead of virtualization? - Can VMs compete with “bare metal” systems?

Categories of Virtualization (1/2)



Categories of Virtualization (2/2)



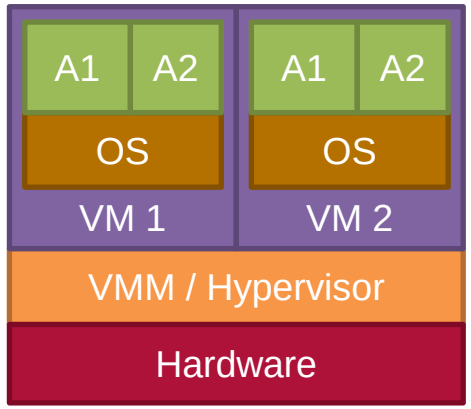
Relevant techniques for Infrastructure as a Services

Overview

- Virtual Resources and Infrastructure-as-a-Service
- **System Virtualization**
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

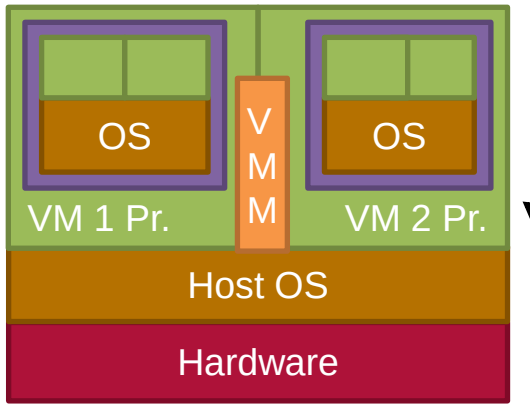
Basic Designs for System Virtualization

- VMM Type I
  - Directly on hardware
  - Basic OS to run VMs
  - Pro: More efficient
  - Con: Requires special device drivers



Also known as bare-metal / native hypervisor virtualization

- VMM Type II
  - VMM as host OS proc.
  - VMs run as processes, supported by VMM
  - Pro: No special drivers
  - Con: More overhead



Also known as hosted virtualization

Conditions for ISA Virtualizability (VMM Type I)

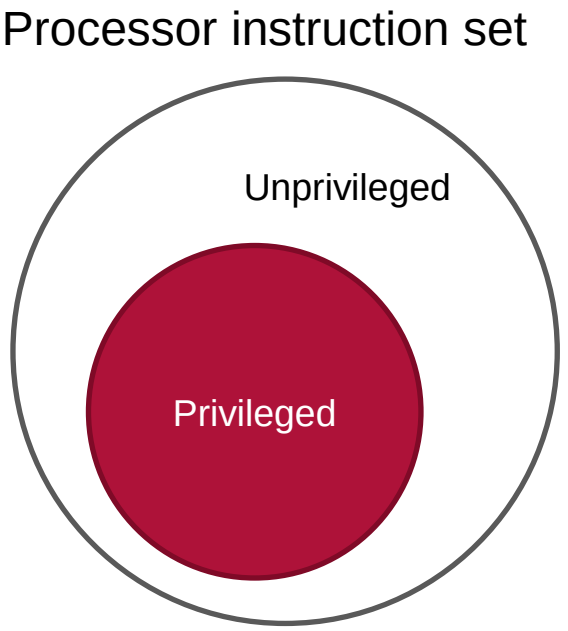
- Fundamental problem for system virtualization:
  - VMM must have ultimate control over hardware
  - Guest operating system must be disempowered without noticing

- Four assumptions in analysis of Popek and Goldberg[3]
  1. One processor and uniformly addressable memory
  2. Two processor modes: system and user mode
  3. Subset of instruction set only available in system mode
  4. Memory addressing is relative to relocation register

Categories of Processor Instructions (1/2)

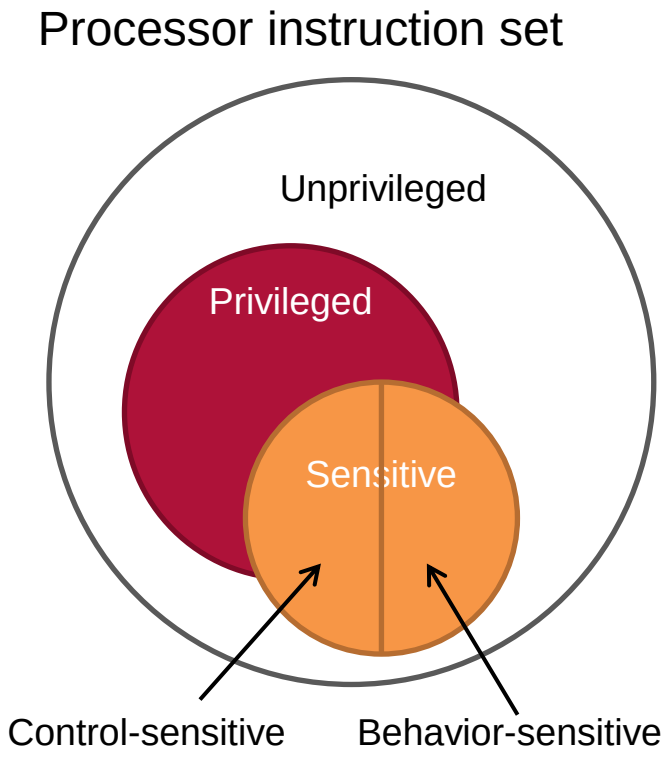
- Privileged instructions
  - Can only be executed in system mode
  - Trap when processor is in user mode

- Examples of privileged instruct.
  - Load PSW (S/370)
    - One bit to indicate system mode
    - Malicious program could modify bit
  - Set CPU Timer (S/370)
    - Defines when user code loses CPU



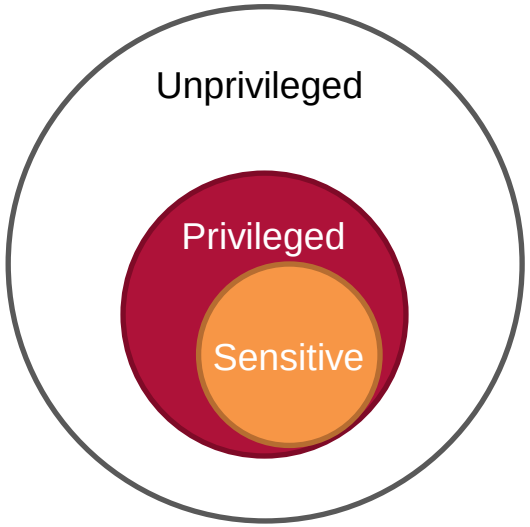
Categories of Processor Instructions (2/2)

- Sensitive instructions
  - Control-sensitive instructions: Change configuration of resources
  - Behavior-sensitive instructions: Behave different depending on configuration of resource
- Examples
  - Load Real Address (S/370)
  - Pop Stack into Flags Register (IA-32)

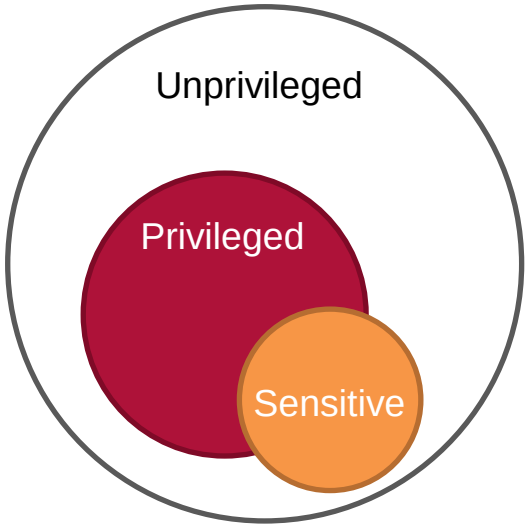


Basic condition for efficient VMMs (Popek and Goldberg‘s Theorem)

“For any conventional third generation computer, a virtual machine monitor may be constructed if the set of **sensitive instructions** for that computer is a **subset** of the set of **privileged instructions**.”



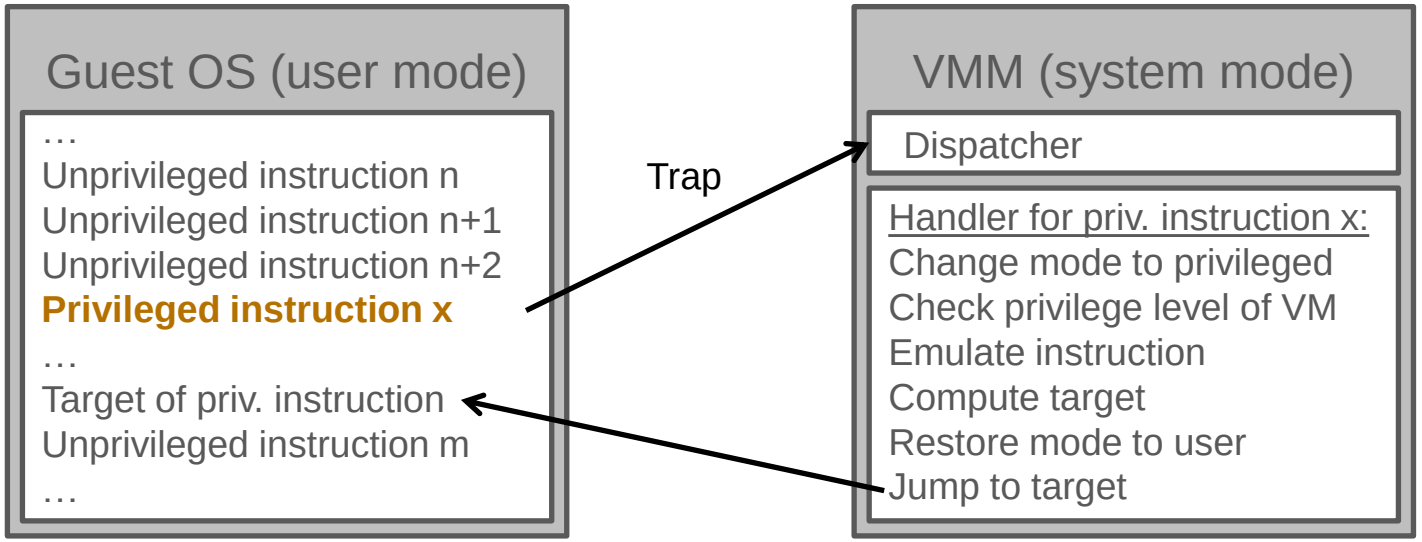
Condition satisfied



Condition unsatisfied

Implications of Popek and Goldberg's Theorem

- Efficient VMM: All non-sensitive instructions run natively on processor
- Trap and emulate: Guest OS calls sensitive instruction
  - Instruction traps
  - VMM emulates instruction operation



Popek and Goldberg‘s Requirements in Practice

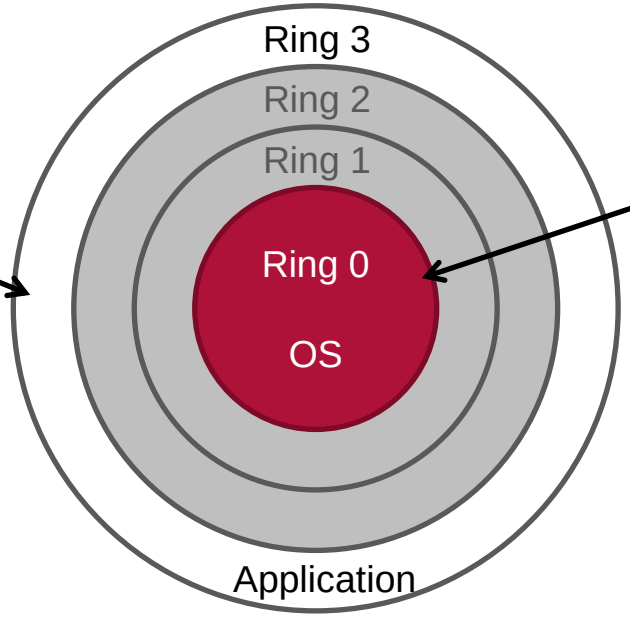
- Which ISAs satisfy Popek and Goldberg’s requirement?
- IBM Power ✓
- Sun Sparc ✓
- Intel IA-32 ✗
  - ~17 critical instructions (= sensitive but not privileged)[4]
  - Critical instructions do not trap, but have different semantics if not executed in system mode

- Apparently, virtualization on IA-32 is possible. So, how can it be done?

IA-32 Architectures

- IA-32 uses rings to manage privileges
  - Four different code privileges possible
  - Designed as generalization of two processor modes
  - To simplify portability, only Ring 0 and 3 used in practice

User space
- Runs applications
- Execution of privileged instruction requires syscall



Supervisor mode
- Full control over CPU
- OS runs in this ring
- Equiv. to system mode

Virtualization of IA-32 Architectures

1. Full Virtualization using Binary Translation

2. OS-Assisted Virtualization

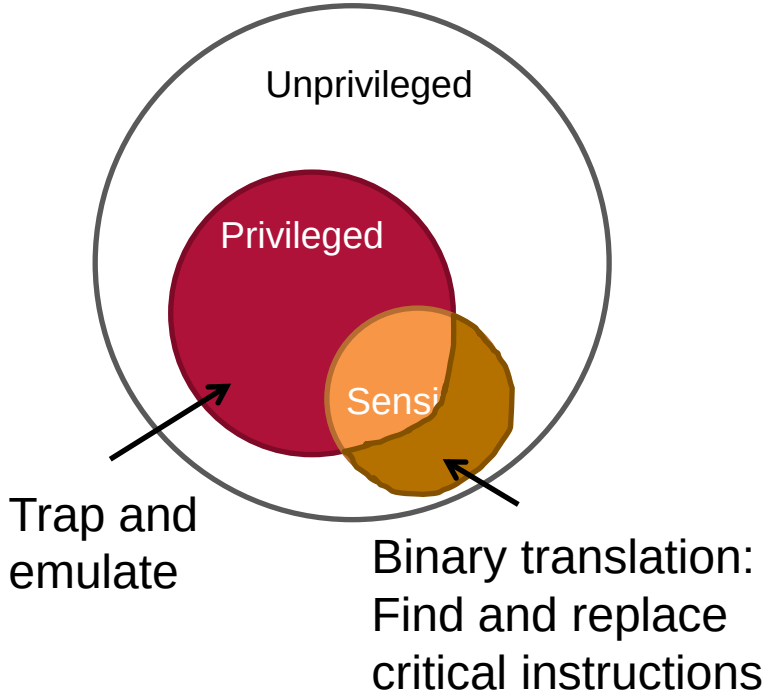
3. Hardware-Assisted Virtualization

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - **Binary Translation**, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

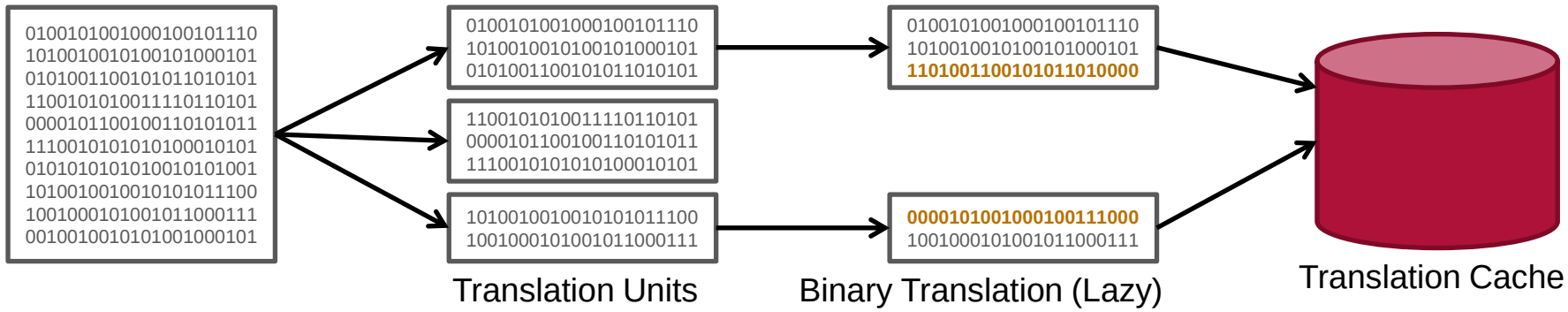
Full Virtualization using Binary Translation

- Idea: Find critical instructions and replace them
  - Run unprivileged instructions directly on CPU
  - Trap and emulate privileged and sensitive instructions
  - Find critical instructions and replace with exception
- Translating a book word for word → inefficient
- Differentiation if critical or regular depends in some cases on the parameters used (e.g. LOAD instruction)
  - Replacement must be done at runtime



Basic Approach for Binary Translation

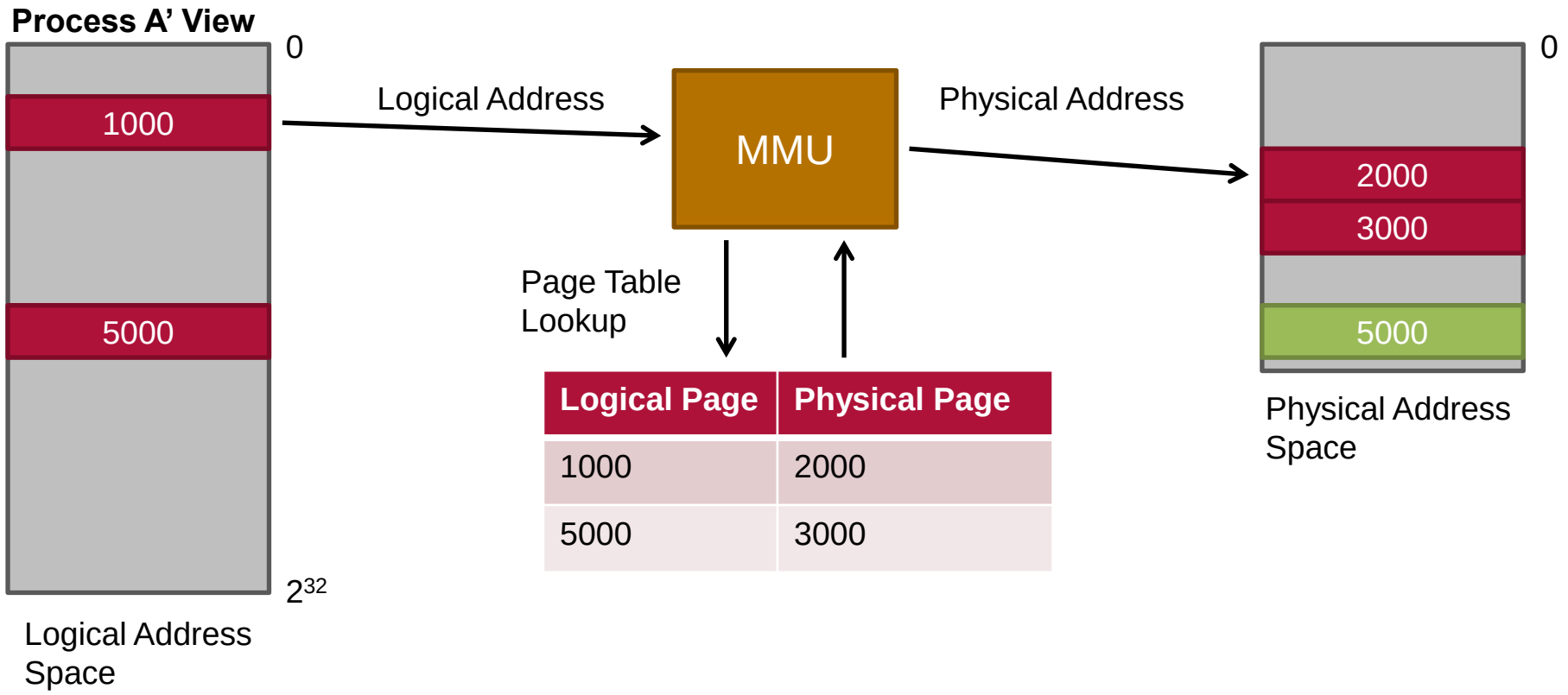
- Separate instruction sequence into translation units
- Check units for critical instructions and modify code
- Modified code is stored in translation cache



- Translation is done lazily
  - Some units may be never translated (exception handling)
  - Frequently used units benefit from translation cache

Memory Management on IA-32 Architectures (1/2)

- MMU translates logical to physical memory addresses



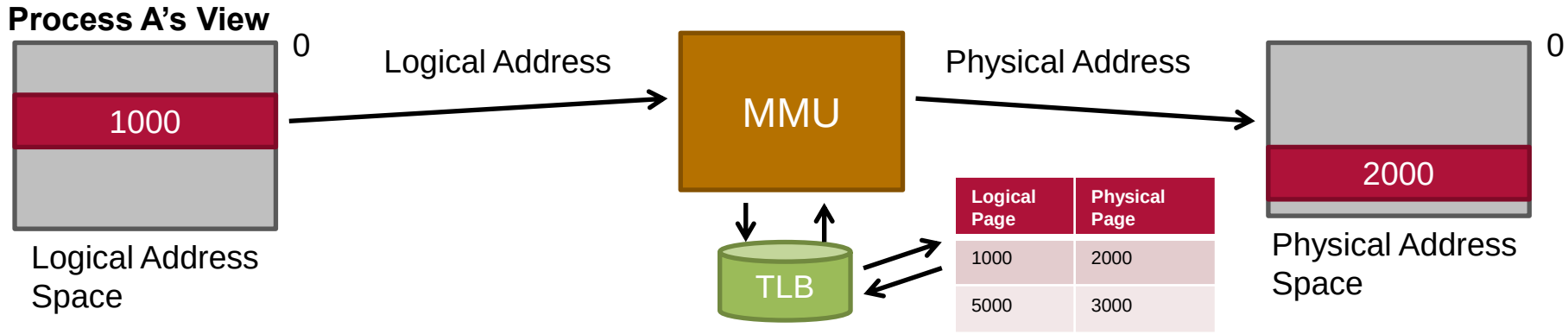
Memory Management on IA-32 Architectures (2/2)

- Page tables are architected on IA-32
  - Hardware knows layout of page table
  - OS modifies the page table, lookup happens transparently
- Page tables reside in main memory themselves → Overhead of memory access essentially doubles
- Idea: Introduce special hardware-accelerated cache to remember recent address translations

  → Translation Lookaside Buffer (TLB)

Translation Lookaside Buffer

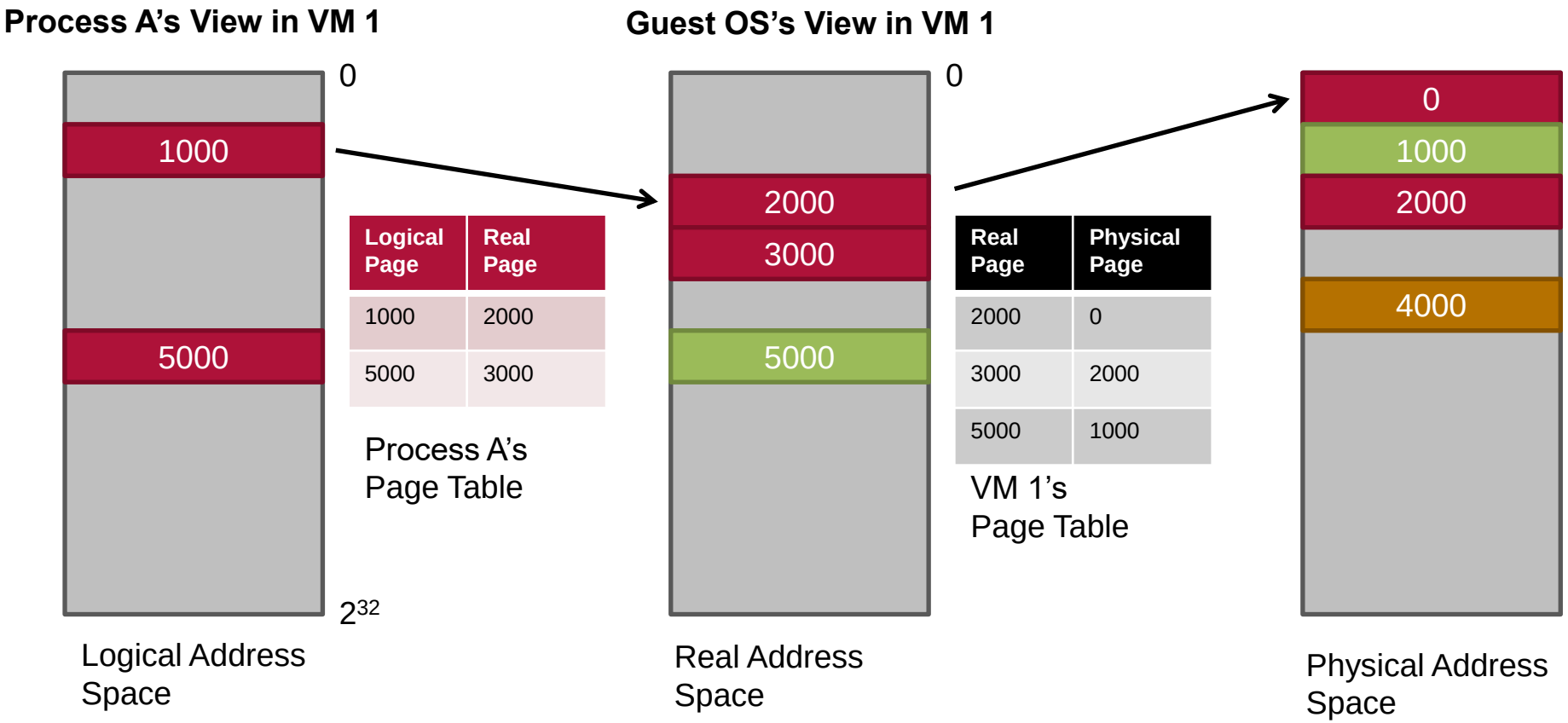
- TLB acts as cache of the MMU
  - fast (~1 cycle hit time)
  - good hit rate (> 99%)



- On IA-32, the TLB is invisible to the operating system
  - Is updated by hardware on every page table lookup
  - Must be flushed on every context switch

Memory Management and Full Virtualization (1/2)

- General idea: Add another level of indirection

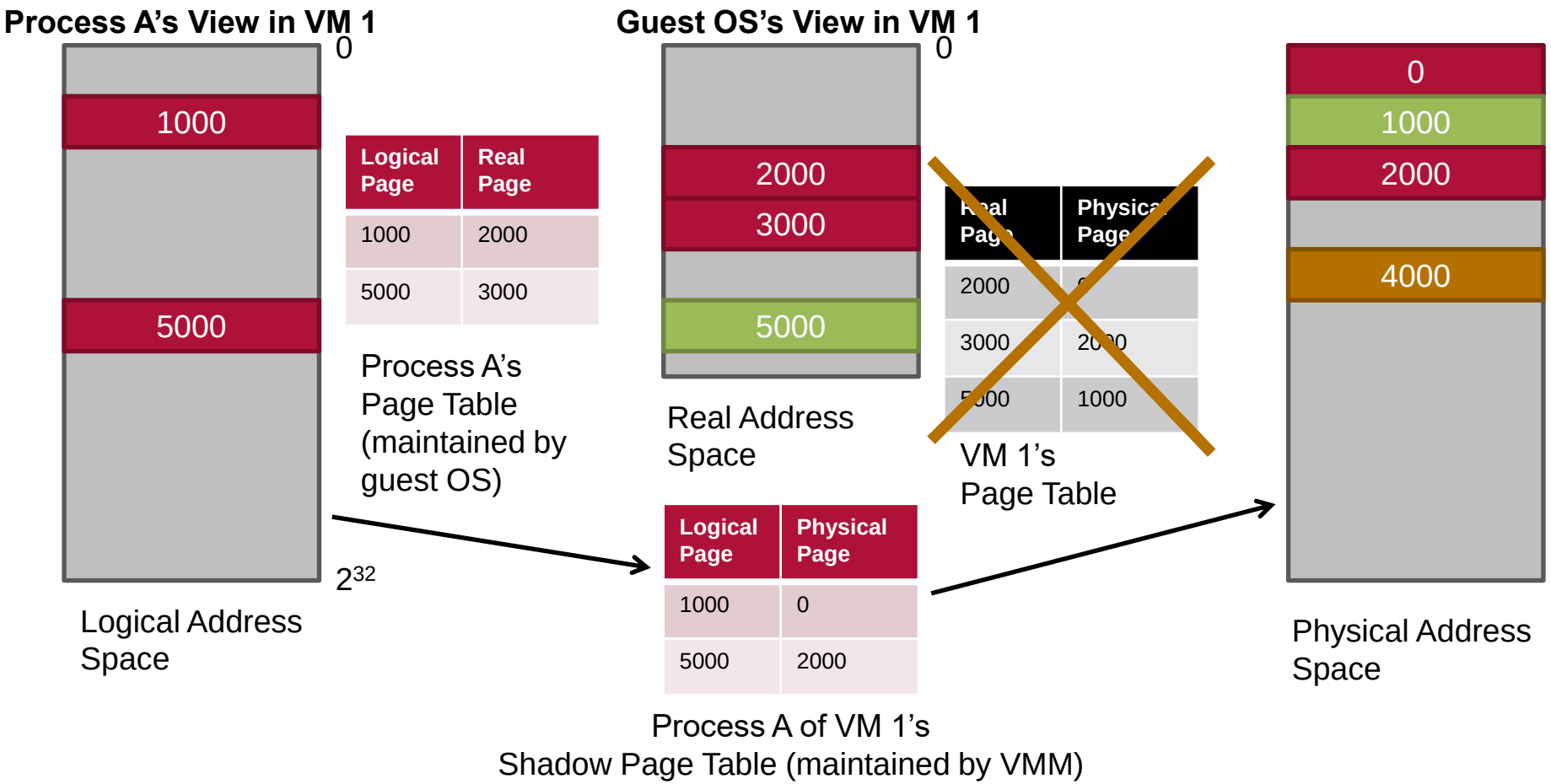


Memory Management and Full Virtualization (2/2)

- Problem: Additional memory access required to resolve address → significant performance decrease

- Practical implementation: ***Shadow page tables***
  - Guest OSs maintain own page tables (for compatibility)
  - But modifications to guest's page table trap and entries are copied to the VMM's shadow page table
  - Shadow page table is then used by hardware
    - Keeps TLB up-to-date
    - Works through virtualization of page table pointer

Memory Virtualization with Shadow Page Tables

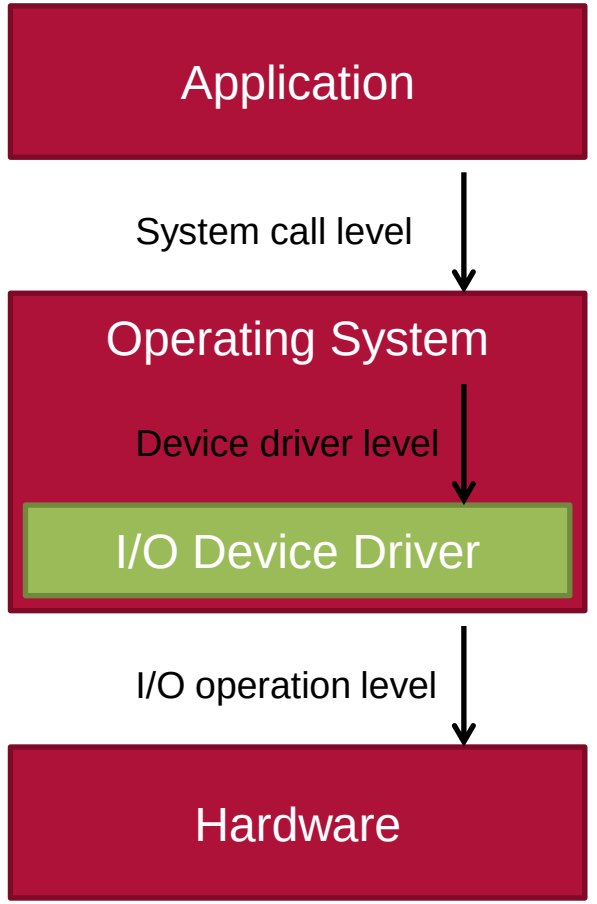


Full Virtualization and I/O (1/2)

- I/O devices can be categorized in five classes
  1. Dedicated devices (e.g. display, keyboard, mouse, …): Not shared among VMs on a very long-time scale
  2. Partitioned devices (e.g. disks): Partitions made available to VMs as dedicated devices
  3. Shared devices (e.g. network adapters): Shared among VMs on very fine-grained time scale
  4. Spooled devices (e.g. printers): Shared among VMs but at higher granularity
  5. Nonexistent physical devices (e.g. virtual NICs): Virtual devices without physical counterpart

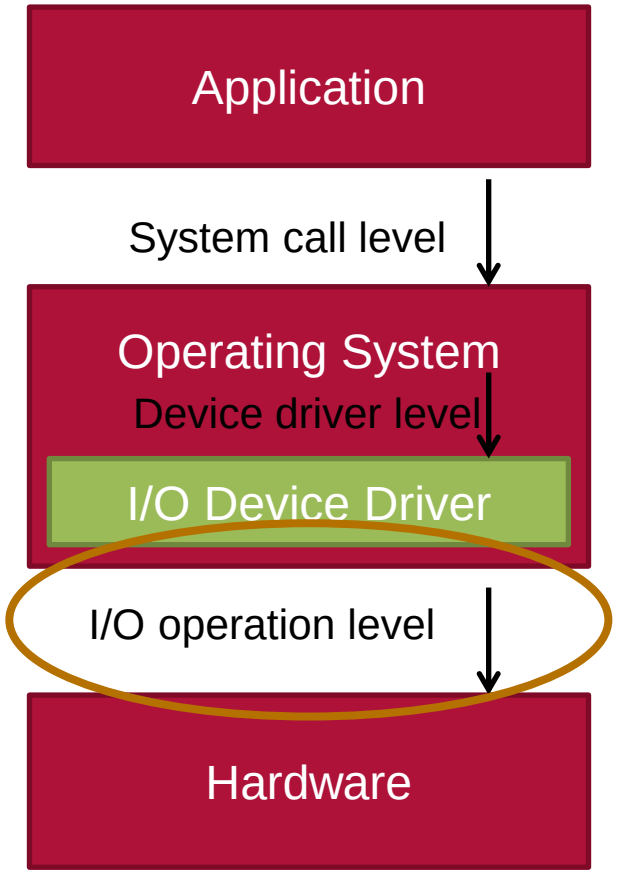
Full Virtualization and I/O (2/2)

- Different levels of I/O virtualization possible
  1. At system call level
  2. At device driver level
  3. At I/O operation level



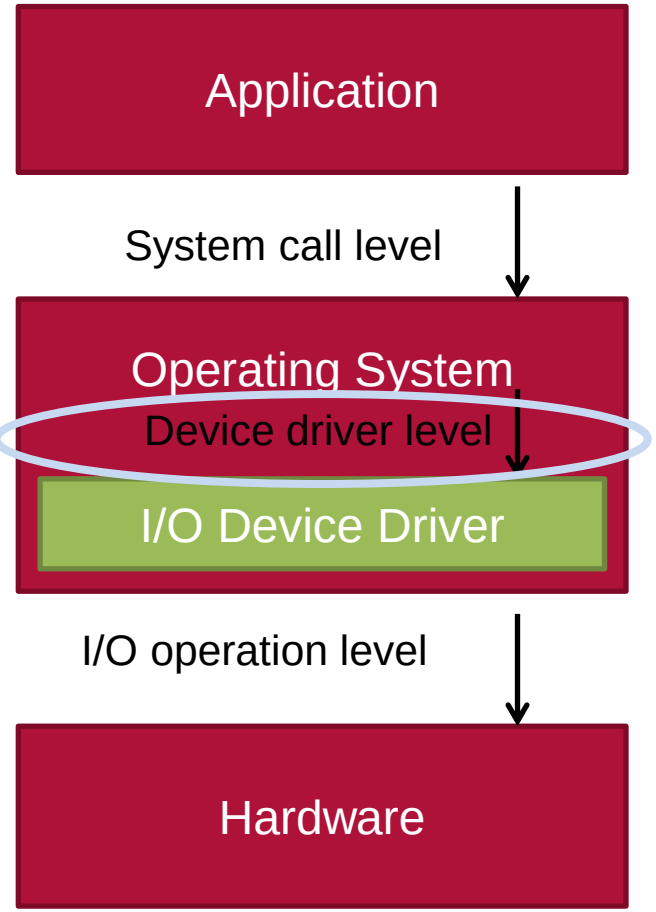
I/O Virtualization at I/O Operation Level

- IA-32 provides special privileged instructions to talk to I/O devices
- Pro: All I/O instructions trap
  - Easy for VMM to intercept them
- Con: Instructions are very low-level
  - Example: Read/write byte to I/O port
  - Higher-level I/O operation consist of several of those instructions
  - Hard for VMM to determine concrete I/O operation, “reverse engineering” required
- Difficult for arbitrary devices



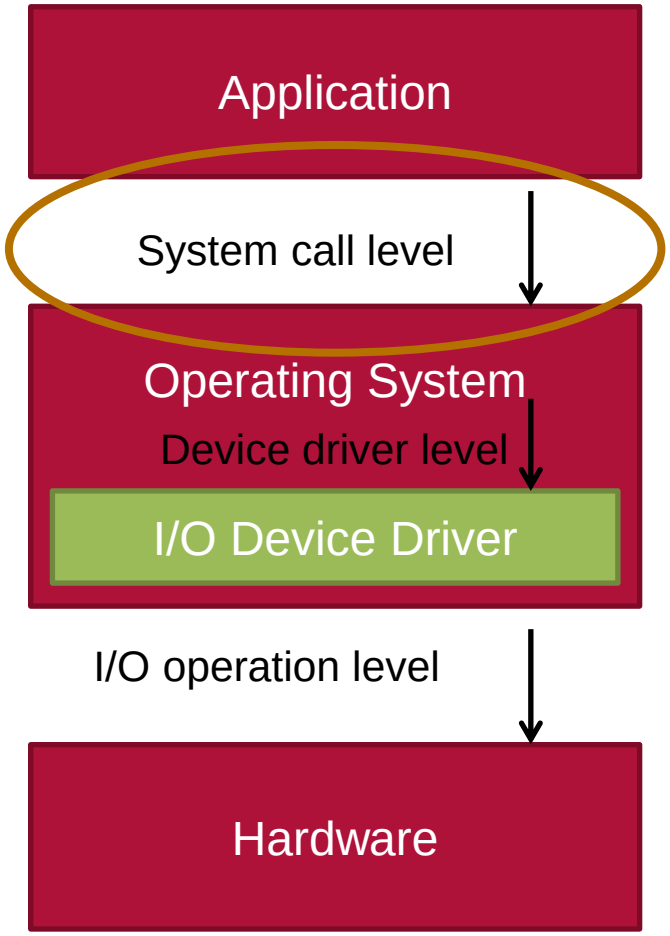
I/O Virtualization at Device Driver Level

- VMM intercepts calls to virt. device driver
  - Converts virtual device information to corresponding physical device
  - Redirects calls to physical device's driver program
- Pro: Natural point for virtualization
  - No "reverse engineering" required
- Con: Requires knowledge of guest's device driver interface
- Not generally applicable, but okay for many practical purposes (e.g. Windows, Linux)



I/O Virtualization at System Call Level

- VMM intercepts system call at OS interface
- Pro: VMM handles the entire I/O operation
- Con: VMM must shadow OS routines available to the user
  - Virtualization must be transparent to the guest
  - Requires broad knowledge of the guest OS's internals
- Very complicated, not seen in practice



Summary Full Virtualization with Binary Translation

- Requires modified guest OS? **NO**
- Requires hardware support? **NO**
- Performance
  - Good approach for compute-intensive applications
    - Unprivileged instructions run directly on CPU
  - Degraded performance for data-intensive applications
    - I/O requires syscalls → privileged instructions
    - “trap and emulate” often requires context switches
    - Context switches lead to flush of TLB

VMware Adaptive Binary Translation

- Modern CPUs are deeply pipelined
- Trapping privileged instructions can be too expensive
- Example: rdtsc (read time-stamp counter), Pentium 4[6]
  - Trap-and-emulate: 2030 cycles
  - Callout-and-emulate: 1254 cycles (Callout method replaces traps with stored emulation functions)
  - In-Translation Cache (In-TC) emulation: 216 cycles
- VMware feature: Adaptive Binary Translation
  - Monitor frequency and costs of traps
  - Adaptively switch between different execution strategies at runtime

Limitations of Adaptive Binary Translation

- Adaptive Binary Translation improves speed over simple “trap and emulate” approach
  - Replaces most traps with faster callouts
  - Some instructions rewritten to run w/o VMM intervention

- However, some limitations remain
  - System calls always require VMM intervention
    - Native system call is ~200 cycles, VMM adds ~2000 cycles
  - Many traps due to shadow page table mechanism
  - Instructions for I/O usually trap, context switch for VMM type II required

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, **OS-Assisted Virtualization**, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

OS-Assisted Virtualization

- Idea of OS-assisted virtualization
  - Make guest OS aware that it is running in a VM
  - Modify the guest source code so that it avoids assistance of the VMM as far as possible

- Denali project also coined term *paravirtualization*[7]

- Requirements for pure OS-assisted approach
  - Source code of guest operating system is available
  - Modified guest OS maintains application binary interface

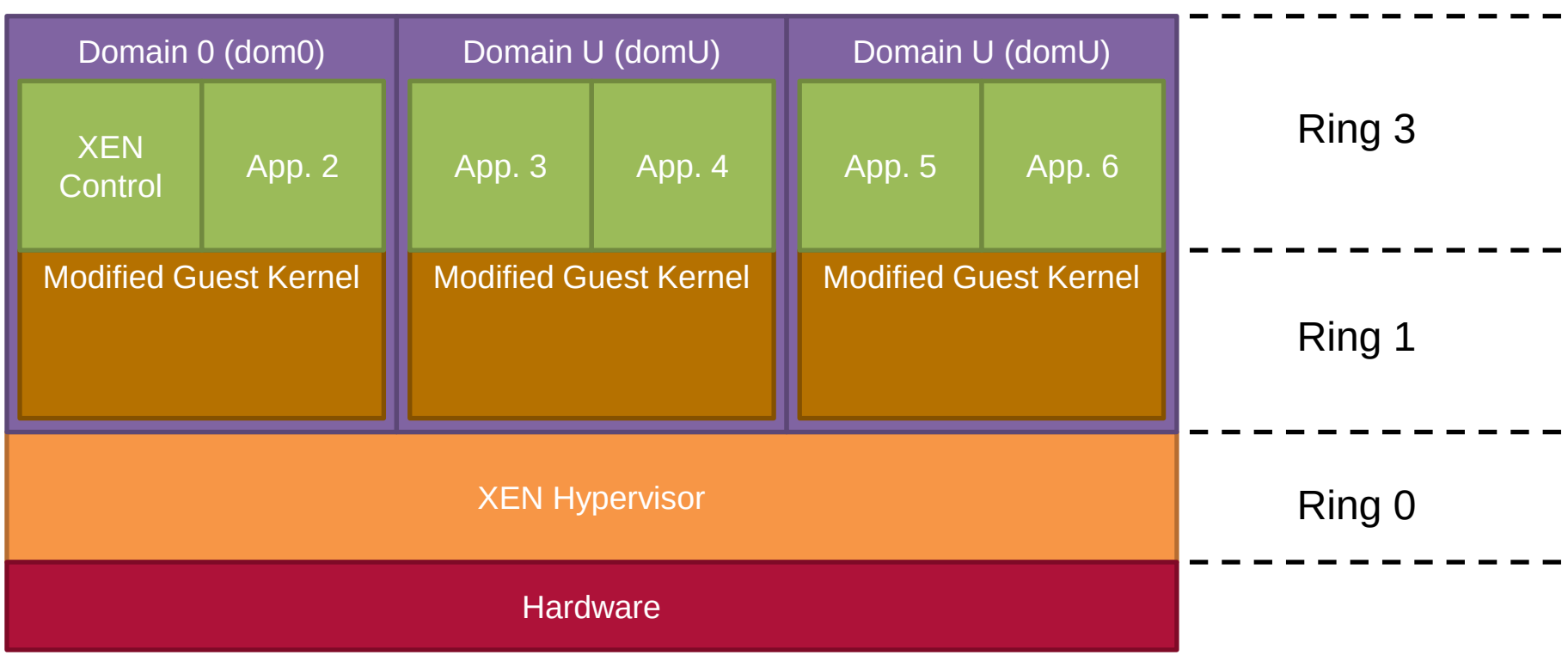
OS-Assisted Virtualization in Practice: The Example of Xen

- Today, most virtualization platforms use OS-assisted virtualization for their device drivers

- Classic representative for OS-assisted virtualization: XEN[8]
  - Type I Hypervisor
  - Available as open-source software
  - Originally developed at University of Cambridge, UK, in collaboration with Microsoft Research Cambridge
  - Presented at SOSP‘03

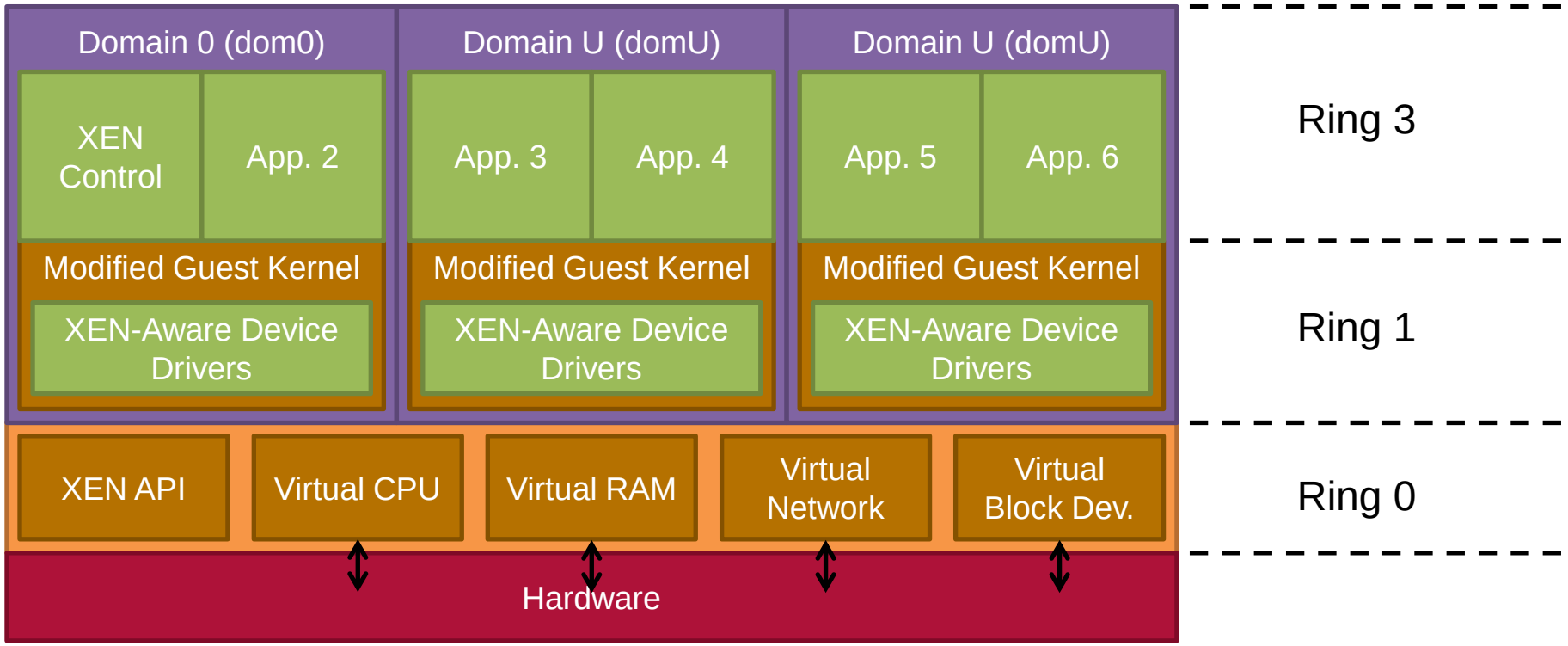
XEN Architecture and Domains

- Domain 0: Privileged guest for control/management
- Domain U: Guest with XEN-enabled OS



Interfaces and Driver Concept (XEN 1.0)

- Communication between XEN and domains:
  - Hypercall: Synchronous call from to domain to XEN
  - Event: Asynchronous notification from XEN to domain



How Does XEN Tackle Full Virtualization Problems?

- Critical instructions do not trap on IA-32
  - Guest OS is aware of virtualization → Critical instructions can be avoided

- Frequent intervention of the hypervisor required
  - Most common reason for required intervention
    - System calls
    - Page table updates
  - XEN cannot get rid of those interventions either
    - Guest domains run in Ring 1
    - However, XEN plays some tricks to decrease frequency

XEN and System Calls

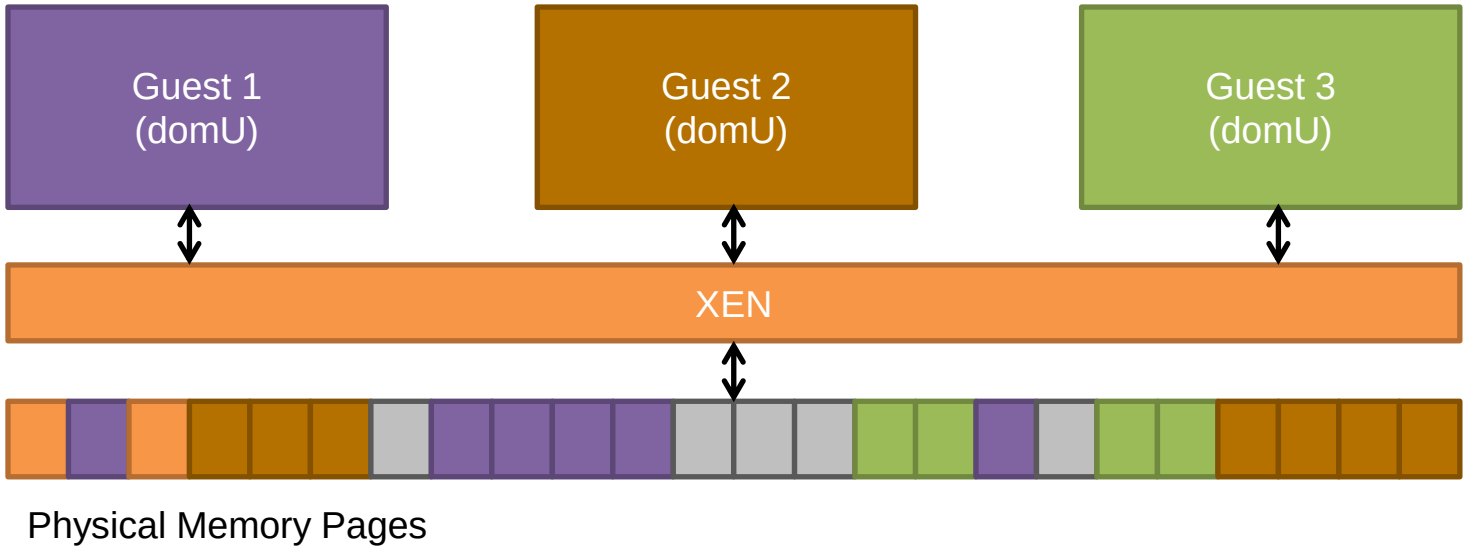
- Major source for VMM intervention with full virtualization

- Syscalls implemented through software exceptions
- Upon exception, hardware consults exception table to find code to handle exception

- XEN allows guest to install “fast” exception handler in the exception table (with automatic forwarding)
- XEN validates the handlers before installing them
- Application can call into guest OS (Ring 1) without indirection through VMM (Ring 0) on each system call

XEN and Physical Memory

- Domain gets fraction of phys. memory at creation time
  - Static partitioning among domains
  - No guarantee partition is contiguous
  - Hypervisor knows which domain „owns“ which pages



XEN and Memory Virtualization (1/3)

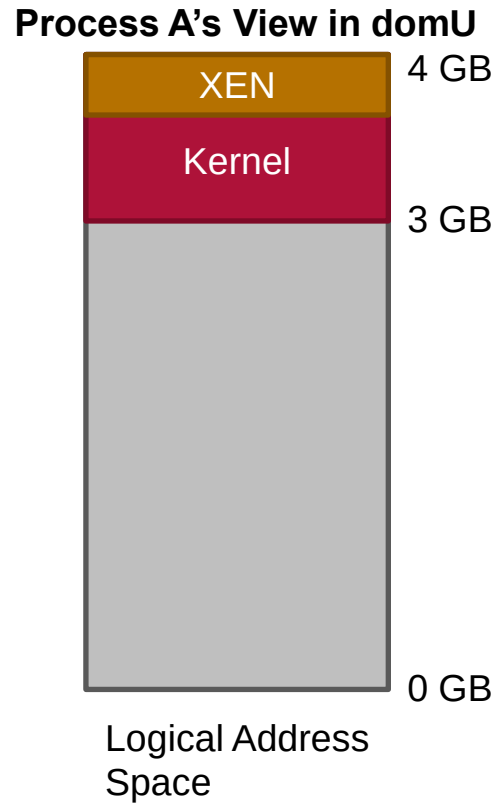
- XEN lets guests maintain their own page tables
  - Guest page tables are visible to the MMU
    - Prerequisite: Guest OS knows its fraction of phys. memory
  - No need for hypervisor intervention on read requests
  - XEN must only validate write requests to ensure isolation

- Procedure for writes
  1. Guest requests page table update via hypercall
  2. XEN checks if mapping address belongs to domain
  3. If ok, allows update to page table

XEN and Memory Virtualization (2/3)

- XEN exists in top 64 MB of every logical address space
  - Kernel can access hypervisor without context switch
  - No TLB flush

- Address region not used by any common x86 ABI → does not break compatibility with applications



XEN and Memory Virtualization (3/3)

- General XEN trick: command batching decreases number of required hypervisor entries/exits

- Example:

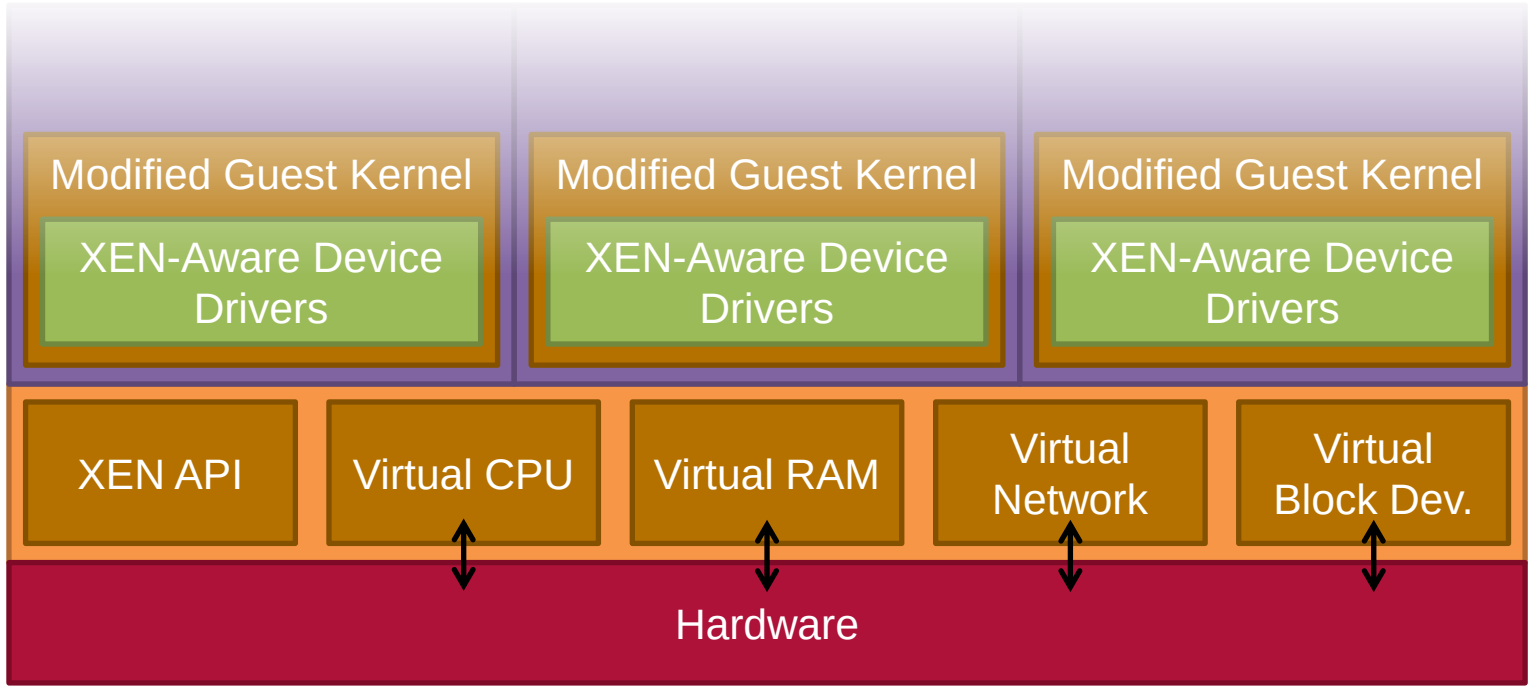
```
void *ptr = malloc(4 * 1024 * 1024); // 1024 4K pages
```

  - Translates to 1024 necessary page table updates
  - Full virtualization: 1024 entries and exits to the hypervisor
  - XEN: Requests collected, submitted with one hypercall
    - Requests are not immediately processed
    - XEN ensures correctness despite delay

  →Only one entry/exit to hypervisor required

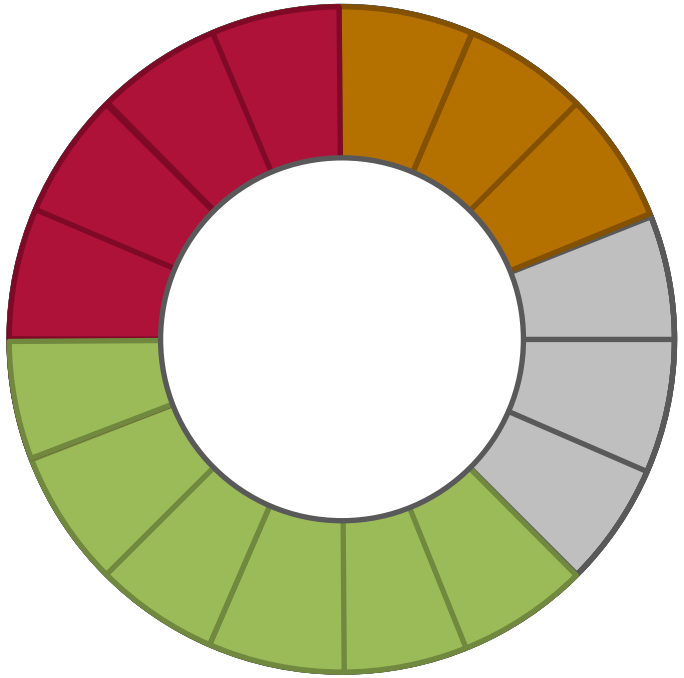
XEN and I/O Virtualization (1/2)

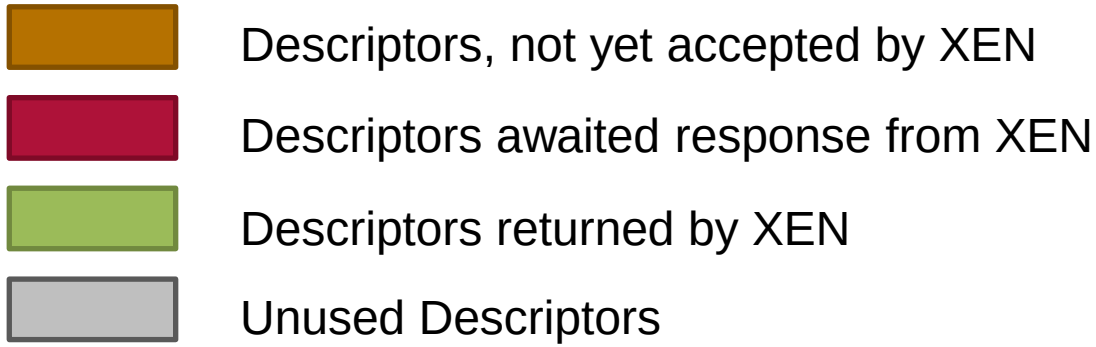
- XEN presents “idealized” hardware abstraction
  - XEN itself contains specific virt. device drivers (XEN 1.0)
  - Domains must only implement lightweight frontend driver
  - Hypervisor and domains cooperate, Communication through hypercalls and events



XEN and I/O Virtualization (2/2)

- I/O data transferred from guests via XEN using shared-memory, async. buffer ring
  - Descriptions reference guest's memory pages
  - Easy implementation of zero-copy transfer





Cost of Porting an OS to XEN

OS Subsection	# Lines Linux	# Lines XP
Architecture-independent	78	1299
Virtual network driver	484	-
Virtual block-device driver	1070	-
Xen-specific (non-driver)	1363	3321
Total	**2995**	**4620**
% of tot. x86 code base	**1.36 %**	**0.04 %**

- No virtual I/O drivers available for XP at that time
- Cost of porting device drivers not considered here

Summary OS-Assisted Virtualization

- Requires modified guest OS? **YES**
- Requires hardware support? **NO**
- Pros:
  - Better performance through cooperation between hypervisor and guest OS
- Cons:
  - Limited compatibility, not generally applicable
  - Increased management overhead for data center operator, different version of OS must be maintained

OS-Assisted Virtualization and Cloud Environment

- OS-Assisted Virtualization is the de-facto standard for I/O virtualization
- All major virtualization solutions provide special drivers
  - VMware, XEN, KVM (virtio project)
- XEN enjoys big support by the community
  - Commercial: Amazon EC2, Rackspace, …
  - Open Source: OpenStack, Eucalyptus, …
  - OS distributors: Linux distributions based on kernel version 3.0 or later
- HW-assisted virt. plays increasingly important role

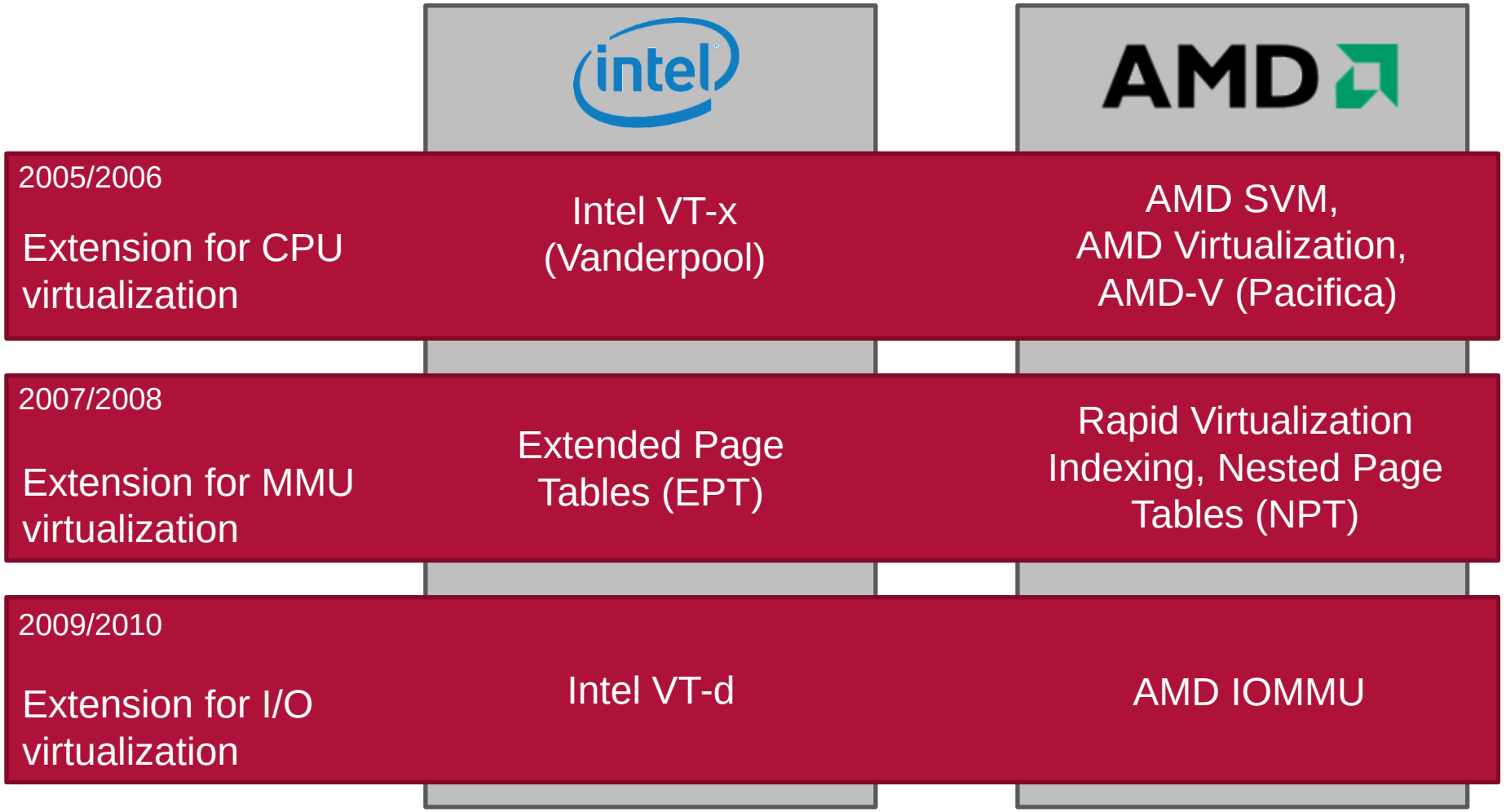
Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, **Hardware-Assisted Virtualization**
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Hardware-Assisted Virtualization

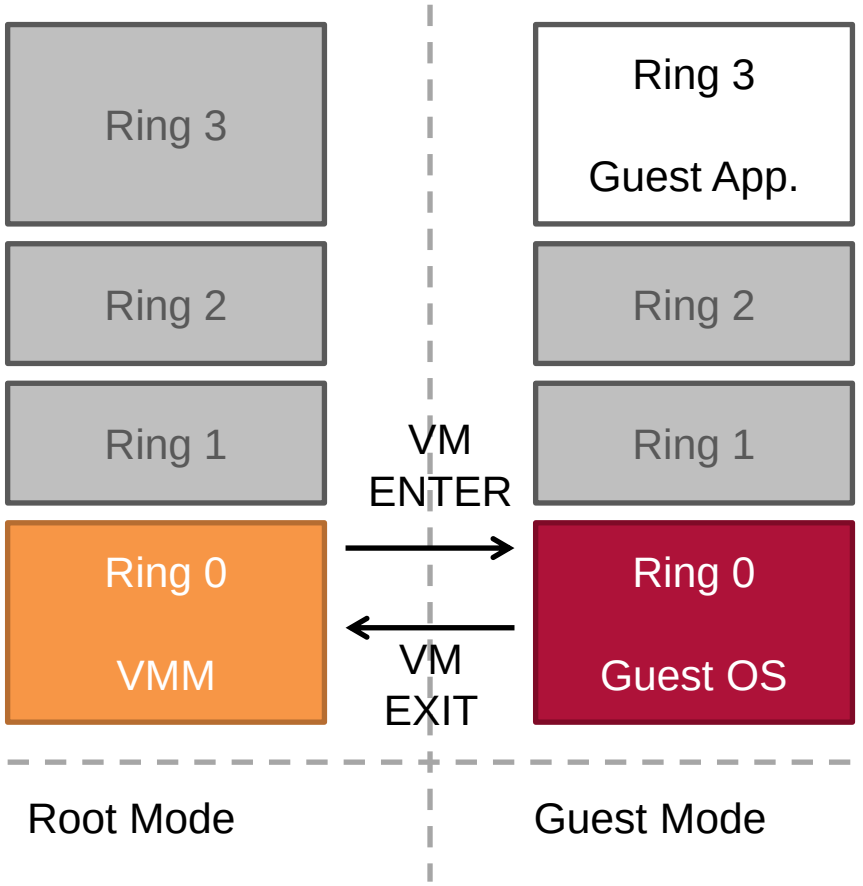
- Most virtualization difficulties caused by IA-32 design
  - Sensitive instructions do not always trap in rings > 0
  - Guests can observe they are not running in Ring 0
- Success of VMware has demonstrated demand for virtualization
- Idea: Extend IA-32 architecture to circumvent virtualization obstacles on the hardware level
  - Independent developments by Intel and AMD
  - Yet, developments share same basic ideas

Incremental Hardware Support for Virtualization



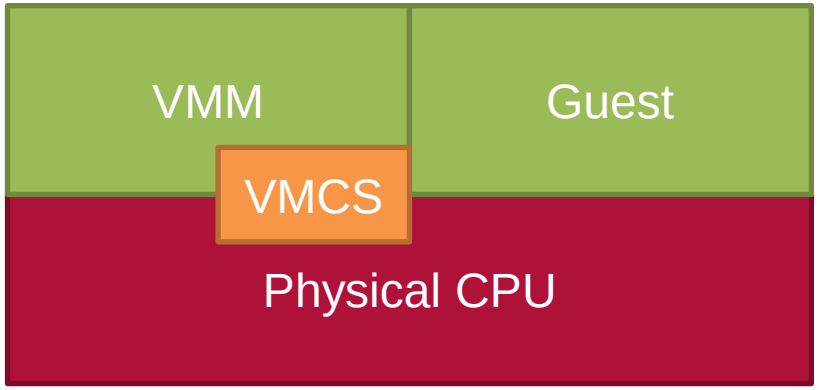
First Generation Support for Virtualization (VT-x, AMD-V)

- Two new CPU modes: root mode vs. guest mode
  - VMM runs in root mode
  - Guest OS in guest mode
- VMM and guest run as "co-routines"
  - VMM can give CPU to guest OS (VM ENTER)
  - VMM can define conditions when to regain CPU (VM EXIT)



VMM Control Structures[11]

- VMM controls guest through HW-defined structure
  - Intel: VMCS (virtual machine control structure)
  - AMD: VMCB (virtual machine control block)



- VMCS/VMCB contains
  - Guest state
  - Control bits defining criteria for VM EXIT
    - Exit on IN, OUT, CPUID, …
    - Exit on write to page table register, …
    - Exit on page fault, interrupt, …
  - VMM uses control bits to “confine” and observe guest

Benefits of 1st Generation Hardware Extension

- VMM controls guest through VMCS in fine-grained way
  - Guest OS continues to run in Ring 0
  - Not all privileged instructions necessarily trap
  - VMM has flexibility to decide which instructions guest is allowed to handle itself
- HW extension eliminates many reasons for VMM intervention compared to classic HW environment, such as syscalls

Limitations of 1st Generation HW Extension

- VMM intervention is still required on several occasions
  - Page table updates
  - Context switches
  - I/O
  - Interrupts

Benefits/Limitations Illustrated (1/3)

```
uint64_t i, s = 0;

for (i = 0; i < 1000000000; i++) {
  s = s + i;
}

printf("s= %ld, c = %ld\n",s, i * (i -1) / 2);
```

Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (1st Gen)
Performance compared to native	95%	95%
Explanation	No VMM intervention, almost all code runs natively	

Only qualitative comparison, don‘t take numbers too seriously

Benefits/Limitations Illustrated (2/3)

```
uint64_t i;

for (i = 0; i < 1000000000; i++) {
  getppid();
}
```

Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (1st Gen)
Performance compared to native	25%	95%
Explanation	System call executed in a ring ≠ 0, VMM intervention required	System call executed in Ring 0, code continues to run natively

Only qualitative comparison, don‘t take numbers too seriously

Benefits/Limitations Illustrated (3/3)

```
uint64_t i;

for (i = 0; i < 1000000000; i++) {
  if(fork() == 0) return;
}
```

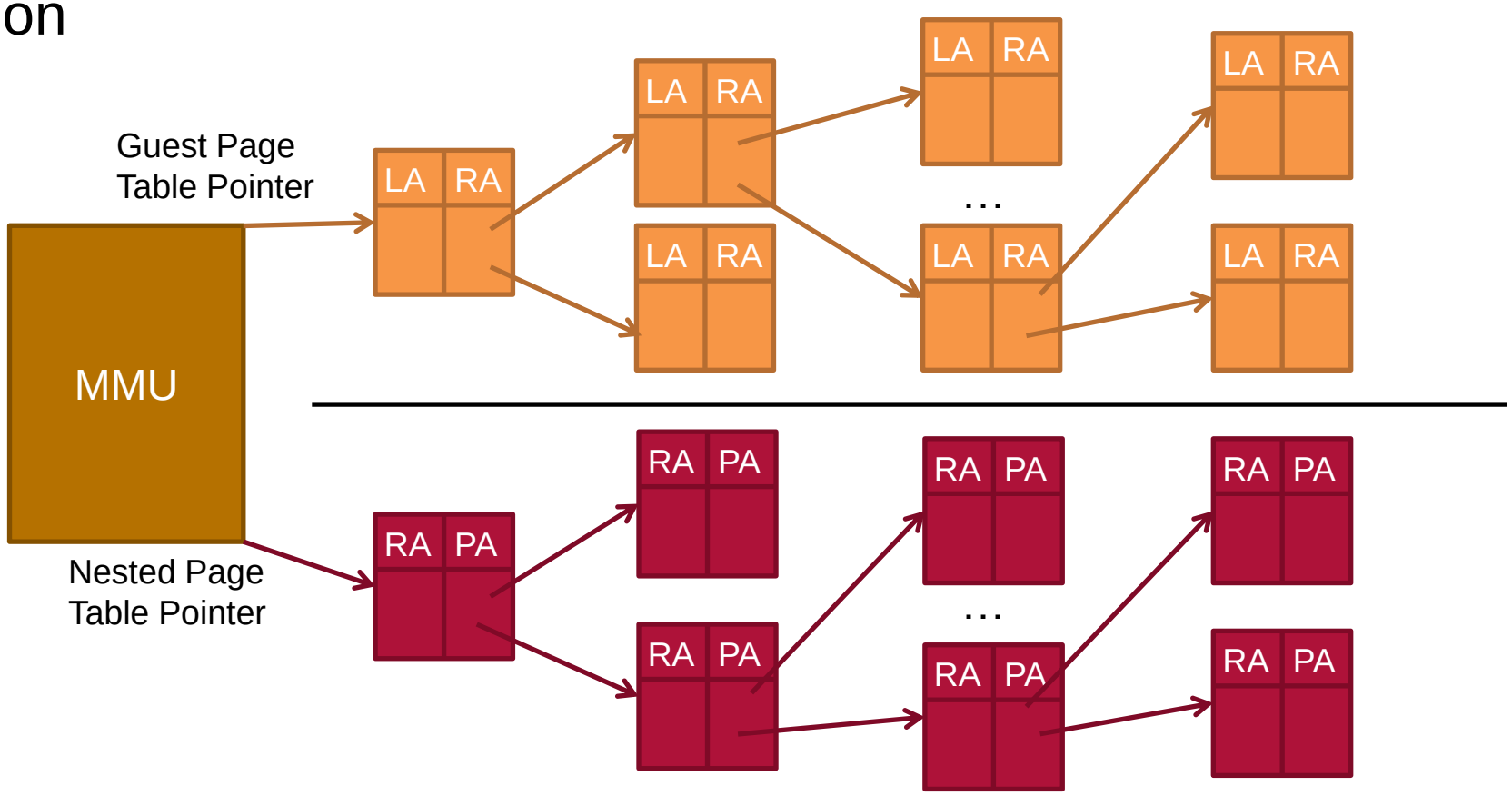
Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (1st Gen)
Performance compared to native	15%	5%
Explanation	Frequent creation of page tables, VMM intervention required due to shadow page tables	

Only qualitative comparison, don‘t take numbers too seriously

Second Generation Support for Virtualization

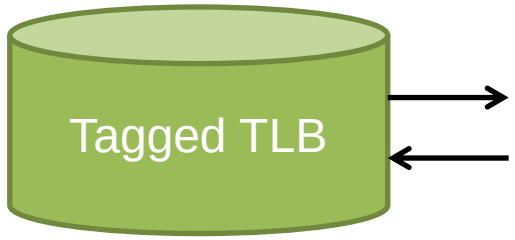
- Extended Page /Nested Page Tables introduce HW support for memory virtualization



LA: Logical Address, RA: Real Address, PA: Physical Address

Tagged Translation Lookaside Buffer

- Translation lookaside buffer continues to cache LA → PA address translation
- Both Intel and AMD introduced tagged TLBs
  - Every TLB entry associated with address space tag
  - Only some entries are invalid on context switch



Address Space	Logical Page	Physical Page
A	1000	2000
A	5000	3000
B	1000	4000

Analysis of EPT/NPT Hardware Extension[11]

- MMU composes LA → RA and RA → PA mapping at TLB fill time
- Benefits
  - Significantly less VMM intervention required
  - No shadow page table memory overhead
  - Better scalability on multi-core CPUs
- Costs
  - High cost for TLB misses: $O(n^2)$, n = page table depth

Limitations of 2nd Gen. HW-Support Illustrated

```
#define S (8192 * 4096)
volatile char large[S];

for (unsigned i = 0; i < 10 * S; i++) {
  large[(4096 * i + i) % S] = 1 + large[i% S];
}
```

Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (2nd Gen)
Performance compared to native	85%	40%
Explanation	Code violates locality assumption, lots of TLB misses, $O(n^2)$ lookup cost	

Only qualitative comparison, don‘t take numbers too serious

Third Generation Support for Virtualization

- Third generation support for virtualization focuses on I/O
- OS-assisted virtualization already decreased CPU overhead for I/O and increased data throughput
  - Cooperation between virtualized device driver and VMM
  - Idealized interface reduced number of VMM interventions
  - However, overhead still too high for high-performance apps
- Goal of hardware support:
  - Performant data transfer between device and guest
  - Isolation between guests

Design Focus of HW-Support: Direct Assignment

- Direct assignment: Guest VM owns a physical device
  - No sharing of device between several VMs
  - Guest VMs run the unmodified device drivers
  - Goal: Efficient I/O without VMM intervention
  - Challenge: VMM must still ensure correctness & isolation

- VMM emulates device
- VMM intervention on every IN or OUT instruction
- Physical devices can be shared across VMs

- VMM provides idealized HW interface
- Guest VM implements driver to idealized interface
- Physical device can be shared across VMs

- Guest runs native I/O driver, "owns" physical device
- Can communicate with device as in native case
- VMM only ensures correctness & isolation

Summary HW-Assisted Virtualization

- Requires modified guest OS? **NO**
- Requires hardware support? **YES**
- Pros: Improved performance even for unmodified guest OSs
- Cons: Reduced flexibility due to hardware constraints (especially for 3rd generation HW support) and specialized hardware required

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - **Virtual Machine Migration**
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Virtual Machine Migration

- Migration: Move VM from one physical host to another
- Motivation:
  - Fault mgmt.: Host reports HW errors, must be shut down
  - Maintenance: Update of BIOS, hypervisor, …
  - Load balancing: Move workload to another physical host

- Desired property: Live migration
  - No shutdown of the virtual machine
  - No disruption of the service
  - Minimal impact for the user
  - Minimize downtime and total migration time

Concerns for Live Migration

- Memory migration
  - Ensure consistency between the memory state of source and destination VM
- Migration of other resources
  - Network resources
    - Maintain all open network connections
    - Do not rely on forwarding of the source host
  - Storage resources
    - Storage must be accessible both at the source and destination VM

Strategies for Memory Migration

1. Push
   - Source VM continues running, sends pages to destination
   - Memory must potentially be sent multiple times
   - Minimum downtime, potentially long migration time
2. Stop-and-copy
   - Source VM stopped, pages are copied to destination VM
   - Destination VM is started after having received all pages
   - Short overall migration time, long downtime
3. Pull
   - Execute new VM, pull accessed pages from source
   - Performance depends on number of page faults

Strategy for Memory Migration in XEN[15]

- XEN pursues pre-copy strategy for memory migration
  - Usage of a push and a stop-and-copy phase
  - Balances short downtime with short total migration time

- Iterative approach: Multiple rounds of push phase, then short stop-and-copy phase in the end

- Similar approach in VMware vMotion: vMotion also capable of slowing down source VM in case memory changes too fast for network transfer

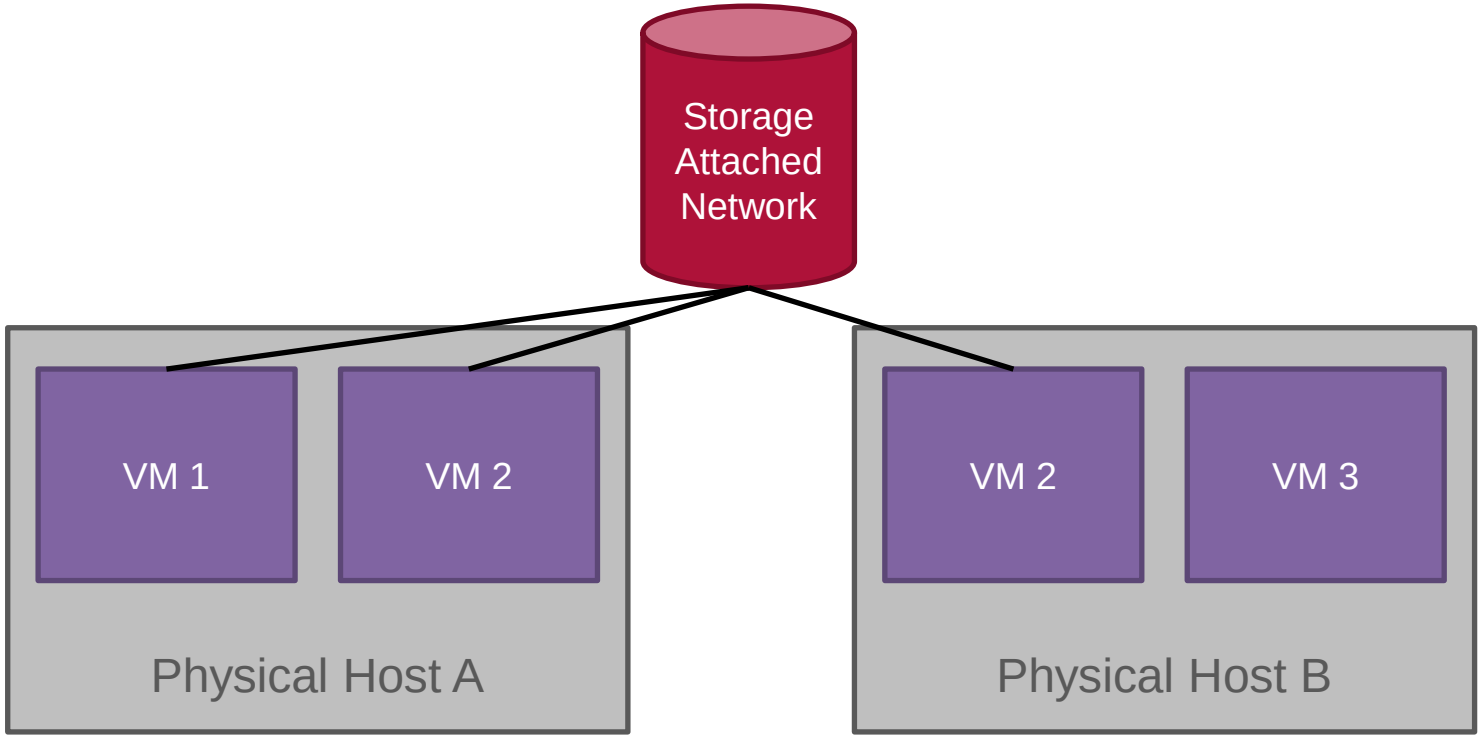
Strategy for Network Migration in XEN[15]

- Assumption: Source and destination VM are on same IP subnet (no IP-level routers involved)

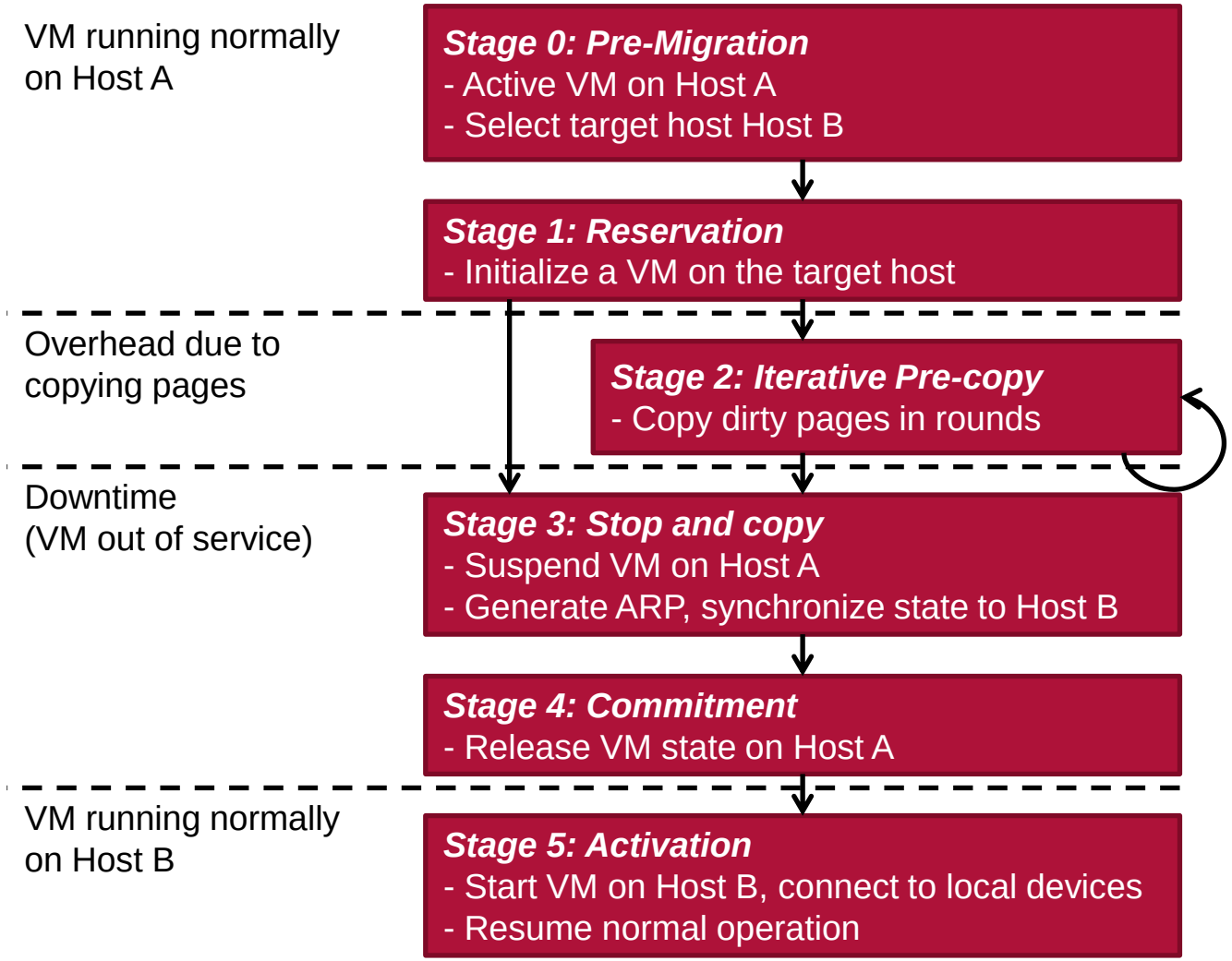
- Approach for migration:
  - Destination VM will have new MAC but old IP address
  - After memory transfer, source host sends unsolicited ARP reply
    - Broadcast message to all hosts on the same network
    - Hosts will remove IP ←→ MAC mapping from caches
  - Upon new ARP request for IP, destination VM will return its MAC address

Strategy for Storage Migration in XEN[15]

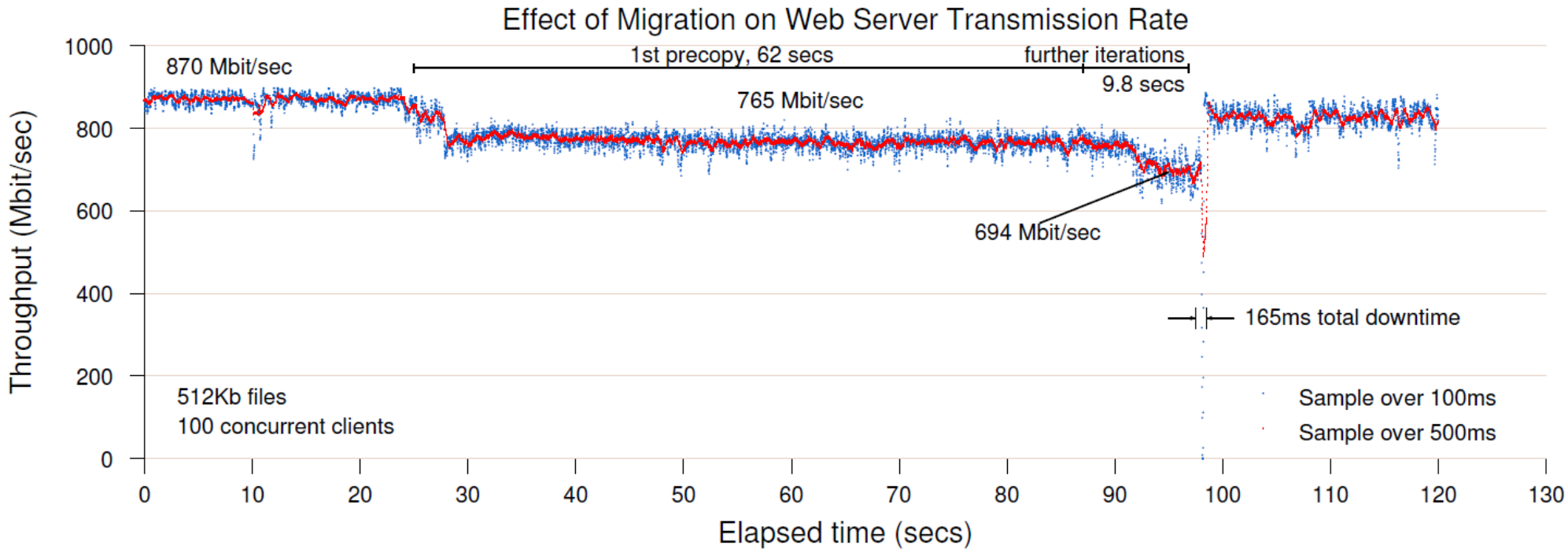
- XEN assumes VMs to reside on storage network (→ virtual / distributed storage)
  - Migration by rerouting network traffic
  - Solved with network migration



XEN Migration Timeline[15]



XEN Migration Performance Figures[15]



- Migration of a running web server VM (800 MB RAM)
  - Web server continuously serves 512 KB file to 100 clients

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - **Resource Isolation and Performance Implication**
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Resource Fairness & Performance Implications

- In commercial IaaS clouds, many VMs often run on the same physical hardware

- Main questions:
  - What notions of resource sharing do cloud operators provide?
  - How is fairness enforced?
  - What are the implications of resource sharing?

Resource Distribution Among VMs

- Storage space: statically partitioned
  - Each VM typically receives predefined fraction of disk
- Main memory: statically partitioned
  - Each VM typically receives predefined fraction of RAM
- CPU: Different methods possible
  - Pinning: Each VM is statically assigned CPU cores
  - Scheduling: VMM dynamically assigns time slots to VMs
- I/O Access: Typically FCFS
  - More sophisticated methods subject to research

CPU Scheduling Algorithms in XEN

- Goals of the schedulers
  - Each VM supposed to receive “fair” share of the CPU
  - High CPU utilization
  - Low response times

- Different algorithms available for XEN, e.g.
  - a general-purpose scheduler,
  - a scheduler optimized for latency sensitive jobs,
  - and experimental real-time schedulers

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- **OS-Level Virtualization**
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

OS-Level Virtualization

- Also known as Container Virtualization
- Motivation: System virtualization comes with too much overhead, large images, and long boot times
- Idea: Do not create virtual machines, instead…
  - Reuse operating system kernel and isolate applications
  - Virtualize access to resources used by processes
    - File system
    - Devices
    - Network
    - Other processes
    - …

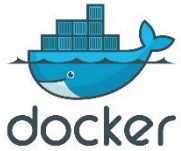
History of OS-Level Virtualization

- Unix v7 **`chroot`** system call (1979)
  - Allows setting the file system root for processes
  - Unix philosophy: almost everything is accessed through the file system → almost everything can be virtualized though **`chroot`**

- Wave of container technologies in early 2000s
  - FreeBSD Jails, Linux VServer, Solaris Zones, …
  - Different degrees of isolation, different tool chains
  - Mostly specific to Linux distributions
  - Never very popular, mainly used by system admins and large companies

History of OS-Level Virtualization

- LXC (2008)
  - User space tools for accessing Linux kernel process isolation features (cgroups, namespaces)
- Docker (2013)
  - Most wide-spread container technology and eco system
- Rocket (CoreOS, 2014)
  - Alternative to Docker, more focused on security and standardization
- LXD (Canonical, 2014)
  - Image-based, focused on entire Linux distributions
- Ubuntu Snap (Canonical, 2018)
  - Universal packaging format for linux distros



Overview

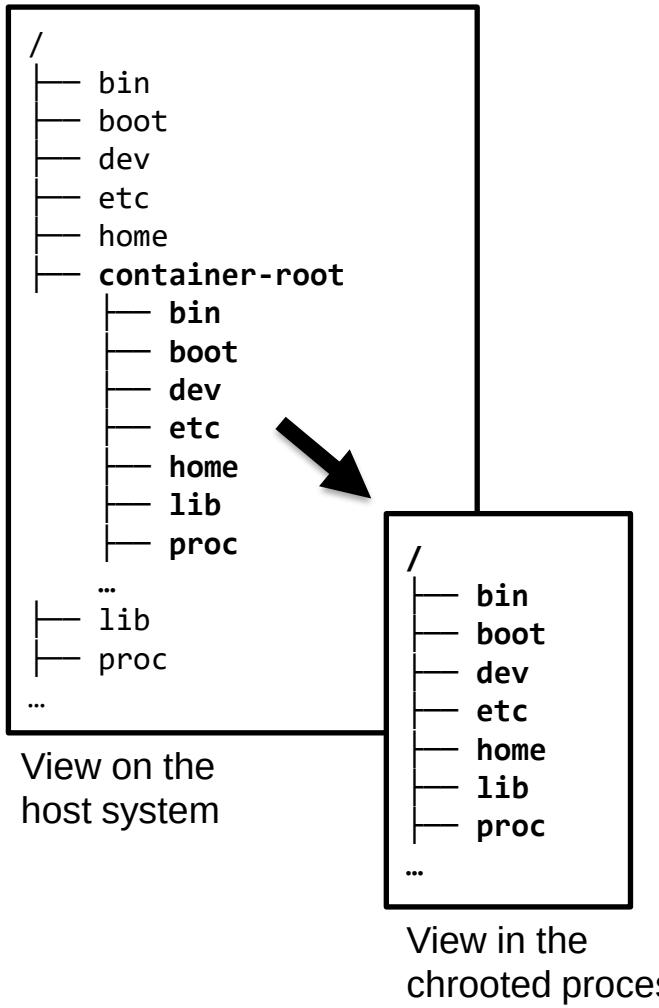
- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - **Linux Containment Features**
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Linux Kernel Features for OS-Level Virtualization

- Linux kernel contains large number of mechanisms for process isolation:
  - `chroot` system call
  - Namespaces
  - Capabilities
  - cgroups
  - SELinux
  - `seccomp`
  - …
- Partially overlapping functionality, but different configuration approaches

chroot

- Oldest mechanism for process isolation
- System call that changes the root directory (“/”) of the calling process
- Since “everything is a file” in Linux, many aspects of the underlying system can be virtualized
- System calls, networking, etc. remain unchanged



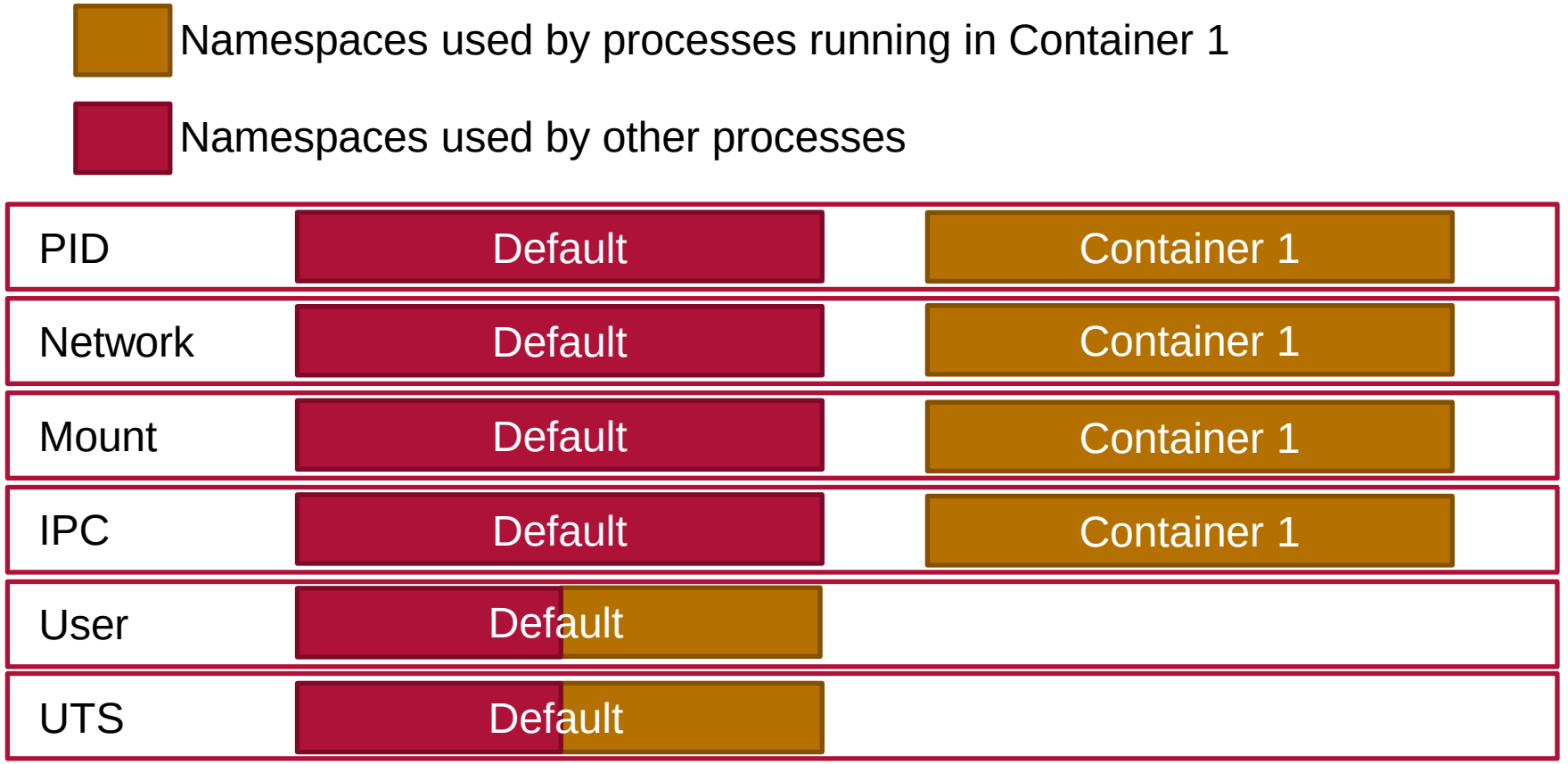
Linux Namespaces

- Provide separate views on kernel resources for processes in different namespaces

- Resources that can be isolated:
  - PID (processes)
  - IPC (Inter-process communication)
  - Network (devices, protocol stacks, firewalls, ports, …)
  - Mount (mount points, file system structure)
  - User (users and groups)
  - UTS (hostname and domain name)

Linux Namespaces

- Container 1: Share users and hostname, but isolate processes, network and mount points

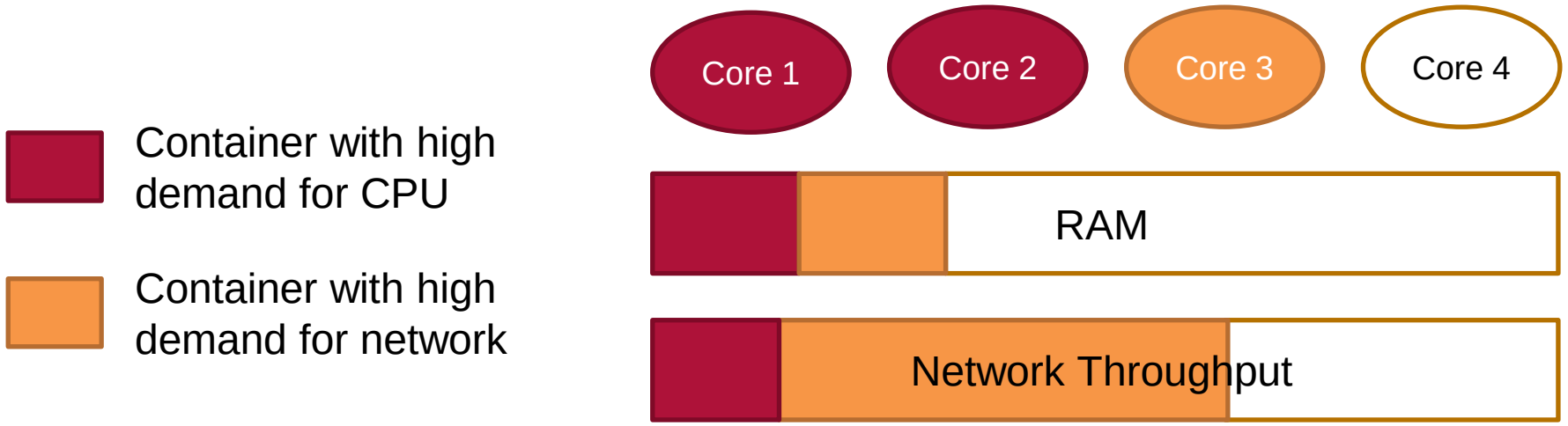


Linux Capabilities

- Traditionally, super user (root) is the only one to perform administrative tasks on a Linux system
- Linux capabilities allow to assign selected rights to new processes
- Long list of possible capabilities, including
  - many system calls,
  - device access,
  - file system modifications,
  - …

Linux cgroups (Control Groups)

- Allow definition of resource usage constraints for parts of the process tree
- Used for limiting CPU, RAM, network, and disk usage for containers



Linux Security Policies

- SELinux, AppArmor
  - Optional security kernel modules
  - Allow rule-based confinement of processes to limit their access to certain files and devices only
  - SELinux: complex, yet powerful rule definitions
  - AppArmor: simpler security profiles

- Seccomp policies
  - `seccomp` system call puts the process in a secure computation mode, where only selected system calls are allowed

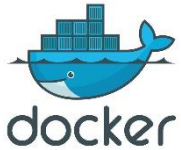
Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - **LXC Containers and Docker**
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Examples for Container Technologies

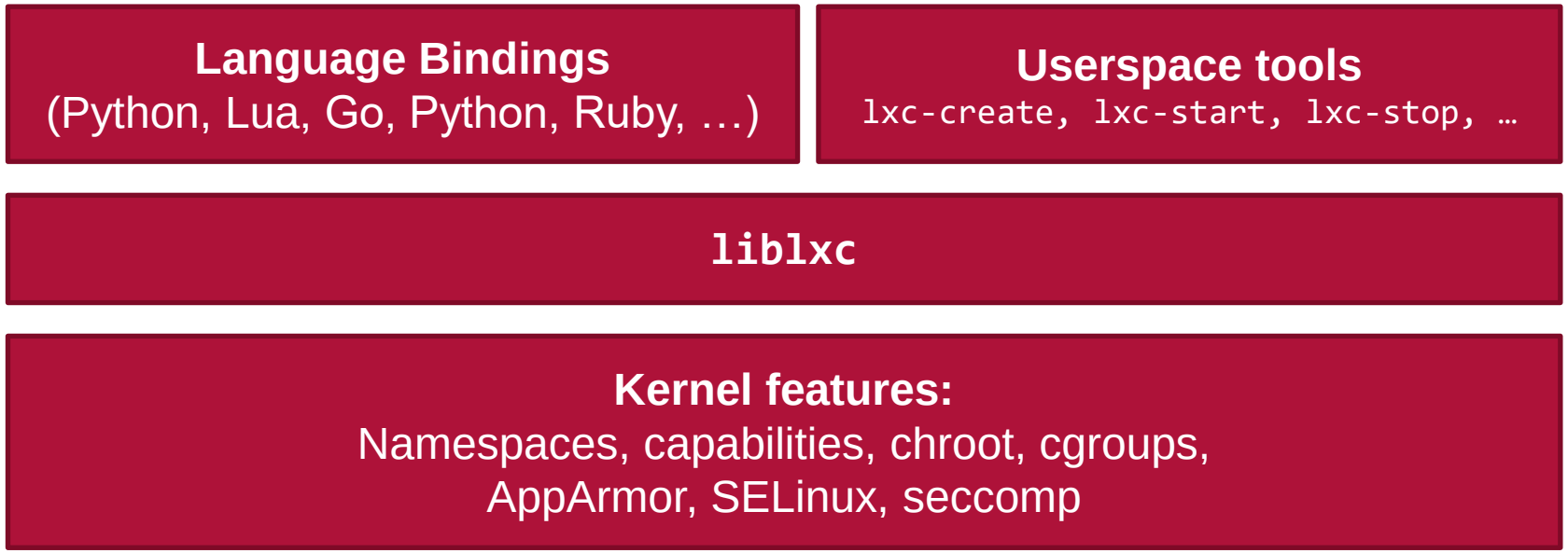
LXC: First container technology widely adopted

Docker: Currently most widely used container platform and eco system



LXC Containers

- LXC: Library and userspace tools (shell scripts) to access kernel process isolation features
- No daemon, instead containers and their file systems are accessible via /var/lib/lxc



LXC Container Creation

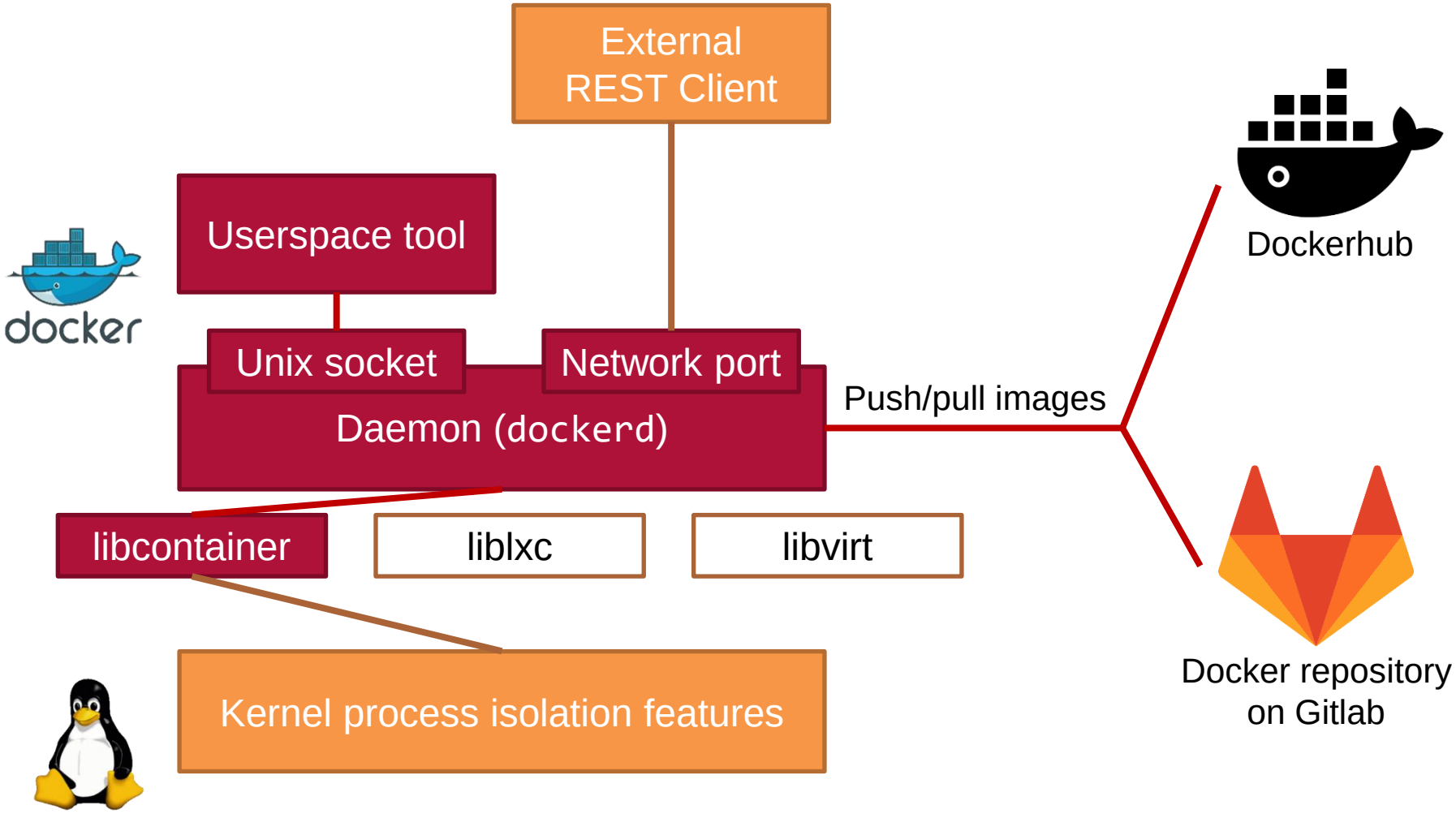
- Container Creation
  1. Container directory allocated under `/var/lib/lxc`
  2. “Template script” executed to populate the file system of the container
  3. Process is spawned and `chrooted` into its file system
  4. Process isolation is configured
  5. Main application process is spawned, all child processes will inherit isolation context
- LXC does not rely on an image format
- No public repository for sharing images (introduced later with LXD)

Docker

- Most popular container technology at the moment

- Docker includes:
  - Daemon and user space tools for managing local containers and images
  - Hierarchical image format
  - Public and private repositories for sharing images (e.g. *Dockerhub*)
  - Build tool for images, called *Dockerfiles*
  - Included and external capabilities for automatic orchestration of container infrastructures

Docker Components

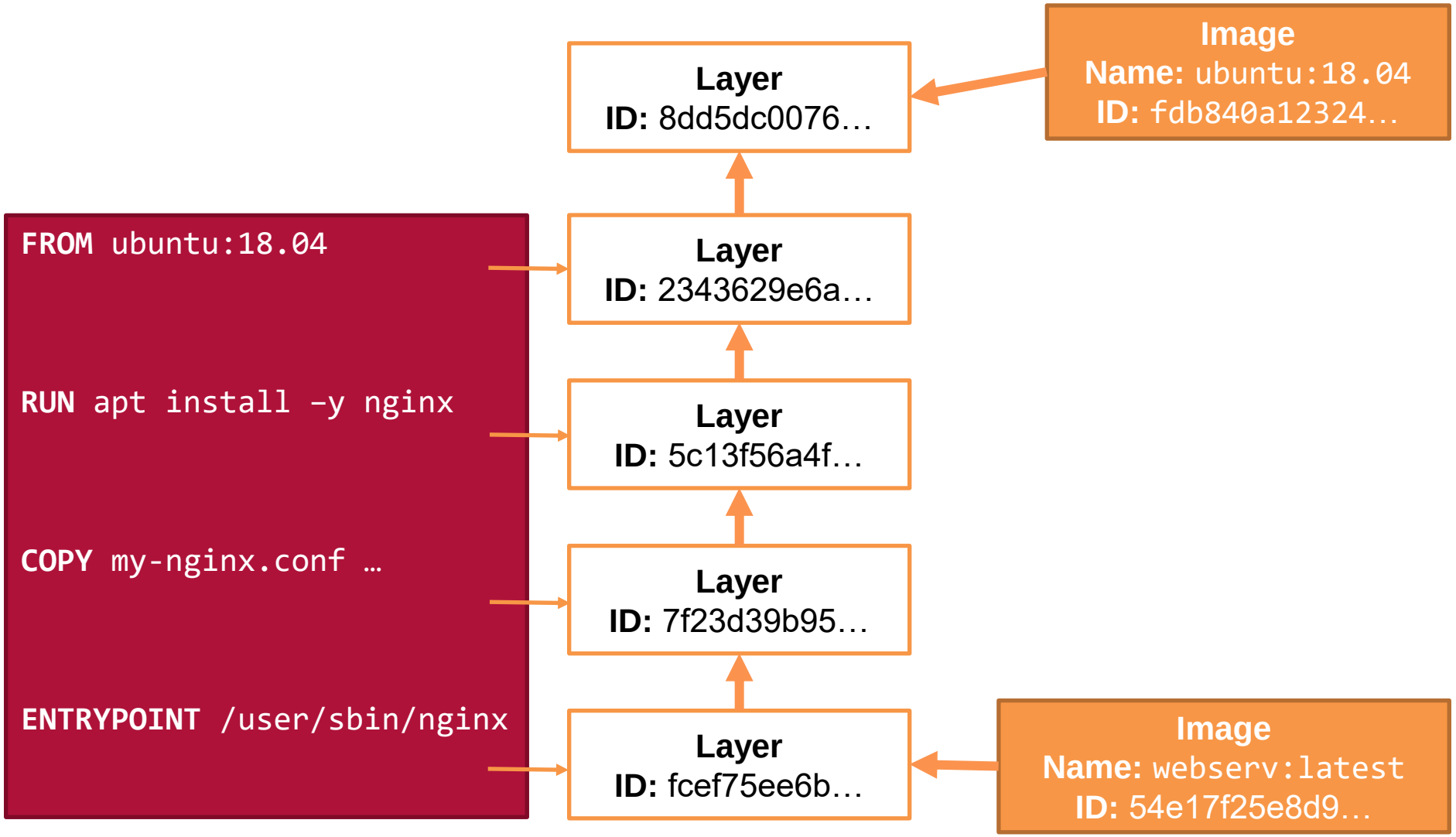


Dockerfiles

- Automatic creation of container images
- Text file with commands that modify files or change configuration values
- Intermediate steps can be cached using hashes, resulting in reduced image build times

```
FROM ubuntu:18.04
RUN apt install –y nginx
COPY my-nginx.conf /etc/nginx/nginx.conf
ENTRYPOINT /user/sbin/nginx
```

Docker Image Format



Docker Image Format

- Docker image contains:
  - List of layers
  - ID (Hash of layer IDs and other config data)
  - Configuration data: ports, mounts, variables, …
  - Other meta info (creation time, author, history, …)

- Each image layer contains:
  - ID of the layer (hash of all files) and ID of parent layer
  - Layer files (tarball): Files that were added/changed on this layer, relative to parent layer
  - Command that was used to create the layer

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - **Comparison to Virtual Machines**
- Case Study: Amazon EC2 / AWS

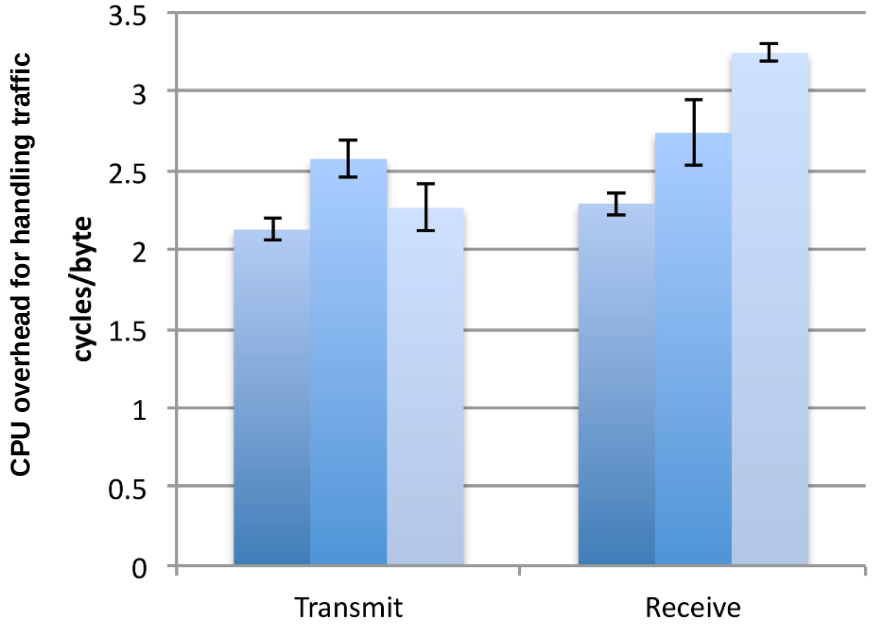
Containers vs VMs: Performance (CPU & RAM)

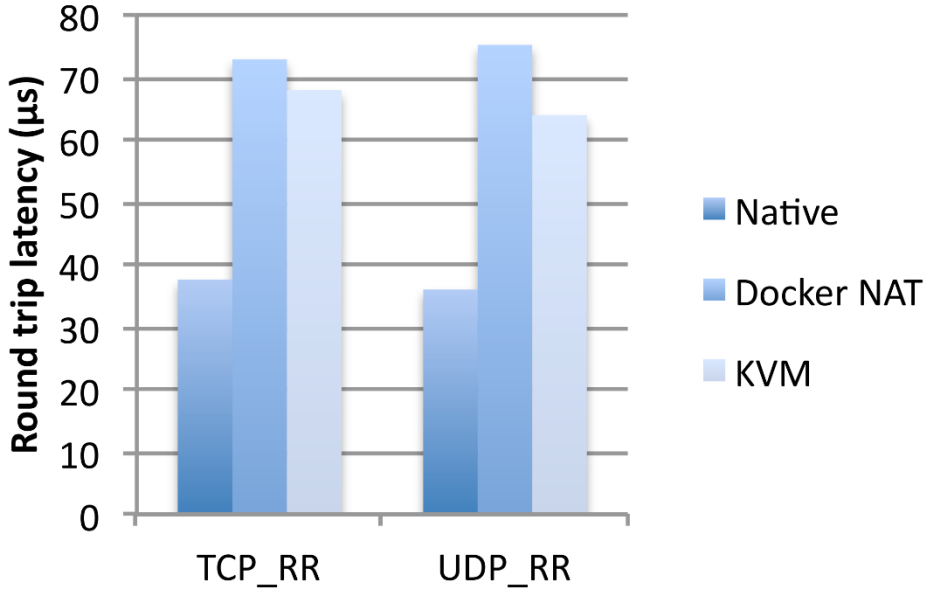
Workload	Native	Docker	KVM
Linpack (GFLOPS)	290.8 [±1.13]	290.9 [±0.98]	284.2 [±1.45]
Memory (Random Access, GIOps/s)	0.0126 [±0.00029]	0.0124 [±0.00044]	0.0125 [±0.00032]
Memory (Sequential Access, GB/s)	45.8 [±0.21]	45.6 [±0.55]	45.0 [±0.19]

- VMs and container introduce low CPU and memory access overhead
  - Prerequisite: exposing cache topology and CPU acceleration features (e.g. NUMA, FPUs, SSE)

Source: IBM Research Report (An Updated Performance Comparison of Virtual Machines and Linux Containers, 2014)
https://dominoweb.draco.res.ibm.com/0929052195DD819C85257D2300681E7B.html

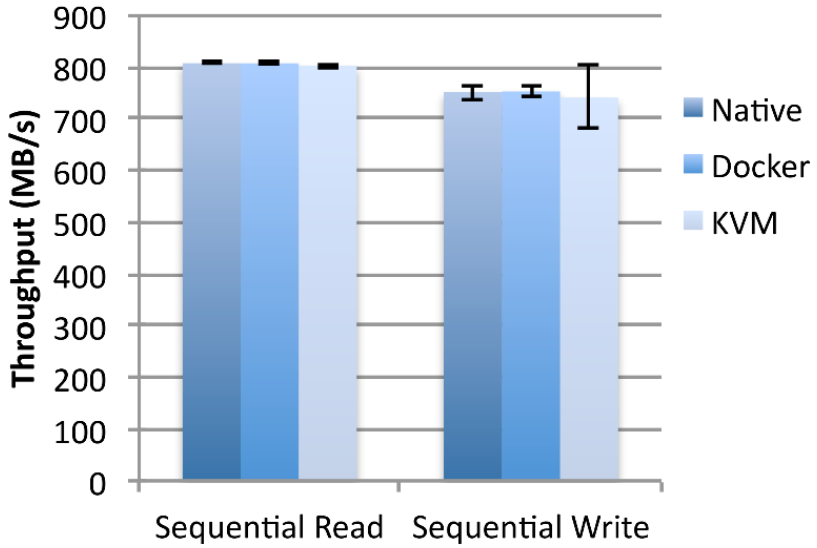
Containers vs VMs: Performance (Network)

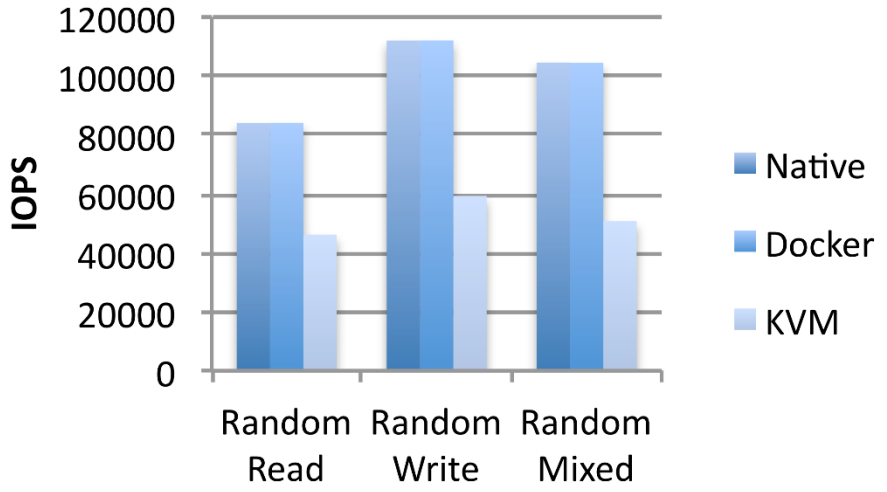




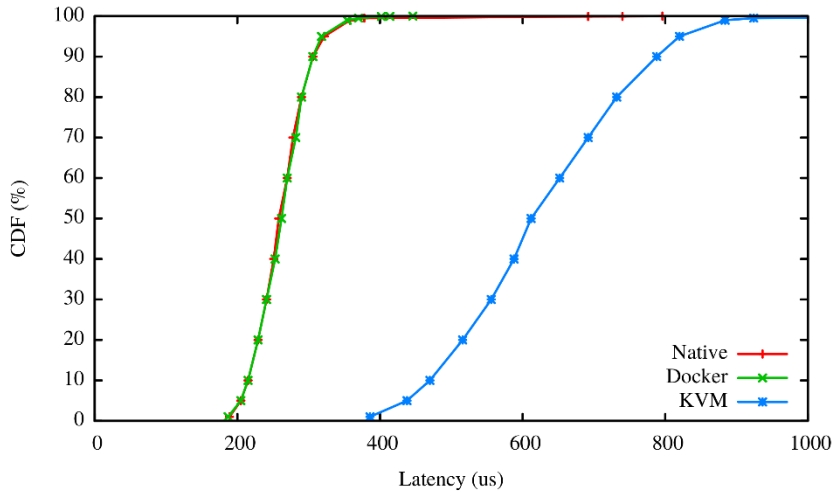
- Low CPU overhead per packet
- Reasons for latency increase:
  - NAT in Docker networking
  - Virtual network device in KVM (NAT might further increase latency)

Containers vs VMs: Performance (Disk IO)





- Similar throughput
- Penalty for latency and IOPs due to virtual IO device



Containers vs VMs: Image Size & Boot Time

- Virtual Machine images usually larger than container images (contain entire OS)
- Boot time of VMs can be orders of magnitude longer than container startup
- Image caching on execution hosts for both VMs and containers

Containers vs VMs: Isolation & Security

- Containers share the host OS kernel
  - Vulnerable to Linux kernel bugs
  - Denial-of-Service attacks: Resource usage, system calls, and context switches of one container can starve others
  - Kernel parameters cannot be tuned to workload

- Virtualization technology older and VMM orders of magnitude less code than OS kernels → more exploits discovered and fixed

Overview

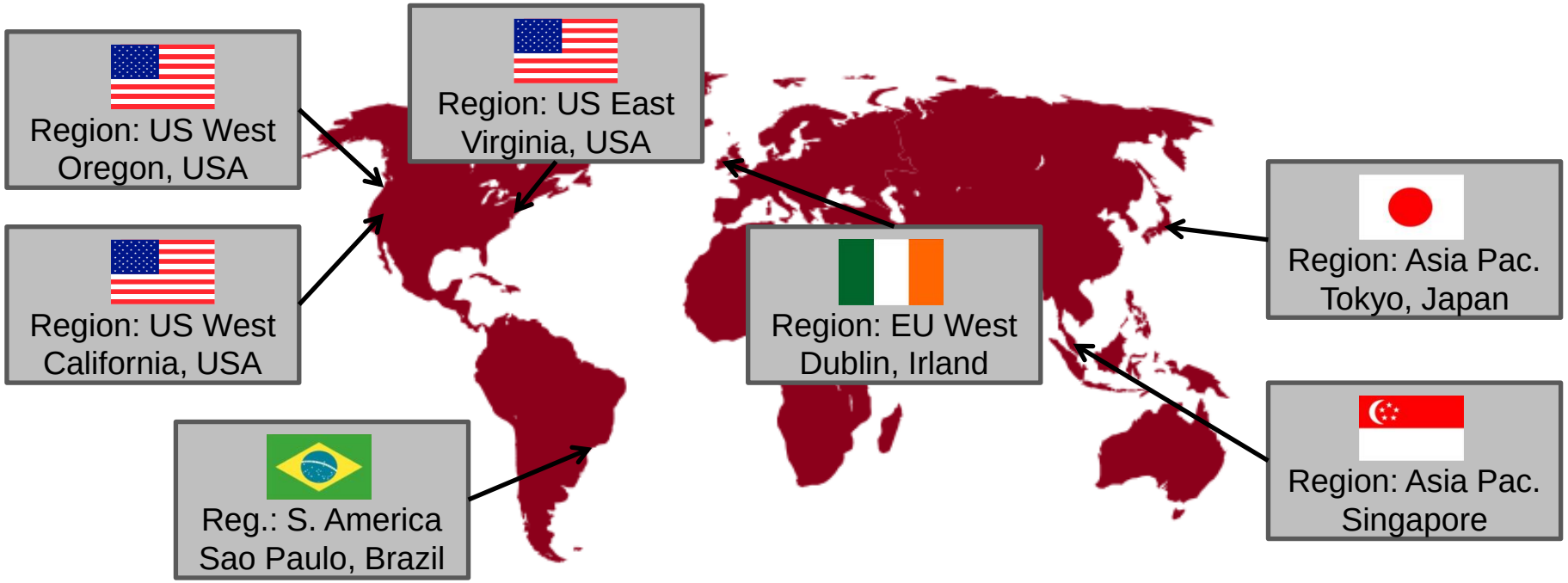
- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- **Case Study: Amazon EC2 / AWS**

Amazon Elastic Compute Cloud (EC2)

- Public IaaS cloud by Amazon Web Services (AWS)
  - Subsidiary of Amazon.com, Inc.
  - Launched in 2006
  - Major cloud platform today

- AWS encompasses increasing number of service
  - Computing: EC2, EMR
  - Storage: S3, Elastic Block Storage
  - Databases: DynamoDB, RDS
  - Analytics: ElasticSearch, Lambda, SageMaker

Geographic Distribution of EC2 Data Centers



- Amazon calls each geographic location a *region*
  - Regions are subdivided into *availability zones*
  - Intra-availability zone traffic is free of charge
  - Per-GB fee for inter-availability zone/inter-region traffic

EC2 Per-Hour Pricing Model

- Per-hour pricing model (hence the term “Elastic”)
  - Amazon charges fee for each started hour of VM usage
  - Customer can shutdown VM at anytime
  - No long-term obligations, reduced risk of over-/under-provisioning
- Concrete per-hour cost depends on several factors
  - Region
  - Virtual machine type (EC2 calls those instance types)
  - Operating system, image (possible license costs)
  - Usage of external services (EBS, Internet traffic, …)

EC2 Instance Types (1/2)

- Instance types define VM classes with particular hardware characteristics

	Small Instance	Medium Instance	Large Instance	XL Instance
Compute power	1 virtual core, 1 comp. unit	1 virtual core, 2 comp. units	2 virtual cores, 4 comp. units	4 virtual cores, 8 comp. Units
Main memory	1.7 GB	3.75 GB	7.5 GB	15 GB
Local storage	160 GB	410 GB	850 GB	1690 GB
Platform	32/64-Bit	32/64-Bit	64-Bit	64-Bit
Price	USD 0.06	USD 0.12	USD 0.24	USD 0.48

Example from region US East (Virginia), Linux/UNIX usage

EC2 Instance Types (2/2)

- "EC2 Compute Unit": Abstract unit for compute power
  - One compute unit corresponds to a 1.0-1.2 GHz AMD Opteron or Intel Xeon of 2007
  - Introduced to improve consistency and predictability among different generations of HW inside data center

- Schad et al. examined performance variations on EC2[18]
  - Instances of same type may be hosted on different generations of hardware
  - Significant performance variations across different instances of same type possible

Containers on EC2

- AWS also has support and lots of options to run containerized applications:
  - Elastic Container Registry (ECR): Store, encrypt, and manage container images
  - Elastic Container Service (ECS): Run containerized applications or build microservices
  - Elastic Kubernetes Service (EKS): Manage containers with Kubernetes
  - AWS Fargate: Run containers without managing servers at all

- EC2: Of course, you can use containers also inside your VMs without any additional Amazon service

Overview

- Virtual Resources and Infrastructure-as-a-Service
- System Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
- OS-Level Virtualization
  - Linux Containment Features
  - LXC Containers and Docker
  - Comparison to Virtual Machines
- Case Study: Amazon EC2 / AWS

Summary: Virtual Resources

- IaaS clouds let customers rent basic IT resources in form of virtual resources
  - Customers have full control over OS and deployed applications in VMs
  - Containers provide alternative or supplement to VMs, isolating individual applications

- Virtualization as fundamental enabling technology
  - Several customers can share physical infrastructure
  - Different techniques to realize virtualization
  - Overhead depends on specific virtualization technique, application, and hardware

References

(1) Andrew S. Tanenbaum, Herbert Bos: Modern Operating Systems, Pearson, 2015

(2) William Stallings: Operating Systems – Internals and Design Principles, 2015

(3) G.J. Popek and R.P. Goldberg: “Formal Requirements for Virtualizable Third Generation Architectures”, Communications of the ACM, 17 (7), 1974

(4) J.S. Robin, C.E. Irvine: “Analysis of the Intel Pentium’s Ability to Support a Secure Virtual Machine Monitor”, Proc. of the 9th Conference on USENIX Security Symposium, 2000

(5) K. Adams, O. Agesen: “A Comparison of Software and Hardware Techniques for x86 Virtualization”, Proc. of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, 2006

(6) A. Whitaker , M. Shaw , S.D. Gribble: “Denali: Lightweight Virtual Machines for Distributed and Networked Applications”, Proc. of the 2002 USENIX Annual Technical Conference, 2002

(7) P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield: “Xen and the Art of Virtualization”, Proc. of the 19th ACM Symposium on Operating Systems principles, 2003

(8) O. Agesen: “Performance Aspects of x86 Virtualization”, VMWORLD 2007

(9) G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig: “Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization”, Intel Technology Journal, 10 (3), 2006

(10) C. Clark, K. Fraser, S. Hand, J.G. Hanseny, E. July, C. Limpach, I. Pratt, A. Wareld: “Live Migration of Virtual Machines”, Proc. of the 2nd conference on Symposium on Networked Systems Design & Implementation, 2005

(11) J. Schad, J. Dittrich, J.-A. Quiané-Ruiz: “Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance”, Proc. of the VLDB Endowment, 3 (1-2), 2010